# Learning to Communicate in Multi-Agent Reinforcement Learning for Autonomous Cyber Defence

# Apprendre à Communiquer entre Multi-Agent en Apprentissage par Renforcement pour une Défense Autonome en Cybersécurité

A Thesis Submitted to the Royal Military College of Canada
Department of Electrical and Computer Engineering
by

Lt(N) Faizan Contractor

Supervisor: Dr. Ranwa Al Mallah

In Partial Fulfillment of the Requirement for the Degree of
Master of Applied Science in Computer Engineering

April 2024

# Acknowledgments

Firstly, I would like to take this opportunity and express my deepest gratitude to my supervisor, Dr. Ranwa Al Mallah, for her unwavering support and guidance throughout this journey. This thesis was extremely challenging and only made possible due to her encouragement and providing a steady source of inspiration along with her knowledge and advice. Next, I would like to thank Dr. Li Li for her timely expertise at stages when the research felt impossible. I also want to acknowledge Royal Military College of Canada for accommodating me with key resources in order to complete this thesis.

# Abstract

Multi Agent Reinforcement Learning (MARL) trains multiple Reinforcement Learning (RL) agents to either achieve a common goal or compete against each other. Popular methods in cooperative MARL with partially observable environments, only allow agents to act independently during execution which may limit the coordinated effect of the trained policies. However, by facilitating the sharing of critical information such as network topology, known or suspected threats, and event logs, effective communication can lead to a more informed decision-making in the cyber battle-space. While a game theoretic approach has shown success in real world applications, its applicability to cybersecurity is an active area of research. The aim of this thesis is to demonstrate the importance and effectiveness of communication between blue agents and to show that relaying key information will allow these agents to stop a malicious actor from compromising hosts across subnets. This thesis also hopes to contribute in the development of techniques that can enhance autonomous cyber defence on an enterprise network. The results demonstrate that through Differentiable Inter Agent Learning, the defender agents play sequential games in Cyber Operations Research Gym and learn to communicate to prevent imminent cyber threats. The tactical policies learned by the autonomous RL agents to achieve the coordination is akin to the human experts that communicate with each other during an incidence response to avert cyber threats.

# Résumé

L'apprentissage par renforcement multi-agents (aussi connu sous le nom de Multi Agent Renforcement Learning, MARL) entraîne plusieurs agents d'apprentissage par renforcement soit pour atteindre un objectif commun, soit pour se faire concurrence. Les méthodes populaires de MARL coopératif dans des environnements partiellement observables permettent uniquement aux agents d'agir de manière indépendante pendant l'exécution, ce qui peut limiter l'effet coordonné des politiques apprises. Cependant, en facilitant le partage d'informations critiques durant l'exécution, telles que la topologie du réseau, les menaces connues ou suspectées et certains événements, une communication efficace peut conduire à une prise de décision plus éclairée en cyberdéfense. Bien qu'une approche fondée sur la théorie des jeux ait fait ses preuves dans des applications réelles, son applicabilité à la cybersécurité constitue un domaine de recherche actif. L'objectif de cette thèse est de l'importance et l'efficacité de la communication entre agents et de montrer que l'échange d'information clé permettra aux agents de la défense d'empêcher un acteur malveillant de compromettre les hôtes à travers les sous-réseaux. Cette thèse vise également à contribuer au développement de techniques permettant d'améliorer la cyberdéfense autonome sur un réseau d'entreprise. Les résultats démontrent que grâce à la technique d'apprentissage différenciable inter-agents, les agents de la défense jouent à des jeux séquentiels dans un gymnasium appelé Cyber Operations Research Gym et apprennent à communiquer pour prévenir les cybermenaces imminentes. Les politiques tactiques apprises par les agents RL autonomes pour réaliser la coordination s'apparentent aux experts humains qui communiquent entre eux lors d'une réponse à une incidence pour éviter les cybermenaces.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

The digital age has brought forward many technological advancements, amongst them are the advantages of global inter-connectivity. However, these benefits also come with increased vulnerabilities to cyber attacks. A cyber attack, defined as an act or an attempt to compromise the properties of confidentiality, integrity or availability of cyber systems, poses a significant threat to organizations as well as individuals [1]. Although the first major attack took place in 1988 with the "Morris Worm", the history of cybercrimes dates back to over a century. In today's times, the "Morris worm" attack is considered very basic as it took advantage of early vulnerabilities in the Unix Operating System [2]. Since then, attackers have grown more sophisticated and covert in targeting organizations and individual users, incurring heavy financial losses and reputational damage. Recent estimates have shown that the average global cost, to organizations, of data breaches reached $4.45 million in 2023 which is 15% increase over the previous three years underscoring the increased challenges in defending cyber systems [3].

To combat the evolving threat landscape, Machine Learning (ML) techniques have gained widespread popularity and among these, Reinforcement Learning (RL) has evolved as a very potent branch in the field of autonomous cybersecurity. In RL, the goal of an agent (learner) is to learn a policy in an environment that maximizes a scalar reward over time and the optimal learned policy allows the agent to make the best decisions (actions) [4]. Combined with deep neural networks, RL has seen a lot of success in complex games like Go [5] and Chess [6], as well as in robotics, and image classification [7]. In cybersecurity, RL has been applied in Network Intrusion Detection Systems (NIDS) that analyze network traffic to detect malware, phishing, and Distributed Denial of Service (DDoS) attacks [8]. However, there are limitations for a single agent to respond to network intrusion especially when the attack is on a larger scale. To address this, Multi Agent Reinforcement Learning (MARL) may be equipped to train multiple agents that can coordinate their actions and collectively respond to threats. This means that the problem can be decomposed into smaller areas of responsibility by multiple agents learning through RL.

MARL can be largely classified as competitive or cooperative. In a competitive setting, agents are trained to compete against one another, whereas in cooperative MARL, agents work towards achieving a common goal. In the latter, communication allows the agents to share observations and coordinate their actions. This leads not only to efficient problem solving but also better decision-making. Moreover, there are different approaches to communication: explicit and implicit. Explicit communication involves the agents exchanging messages via a communication channel, while implicit communication relies on the agents observing each other's actions to infer their intentions [9].

This research delves into the potential of explicit communication between agents in cooperative MARL for Autonomous Cyber Defence (ACD). ACD is defined as a "terminology focusing on the automated decision-making agents for cyber systems (like enterprise network, industrial control systems) to mitigate highly complex cyber attacks [10].” Furthermore, this study demonstrates the effectiveness of communication among defender agents in a simulated network environment and aims to contribute to the development of advanced techniques that can help in protecting enterprise networks against cyber threats. The application of MARL, coupled with explicit communication, offers a different avenue for augmenting ACD capabilities, mirroring the tactics used by the human experts in navigating cyber battlespace.

## 1.1 Motivation

Organizations employ many techniques and systems such as Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), penetration testing, and more as detection and prevention measures against cyber attacks. These techniques require human intervention to some extent and can lead to delays in detection and response. With novel and automated attacks, it has become increasingly difficult for analysts to cope with the volume and complexity of these incidents. A solution to these limitations is to automate some of these techniques which can lead to efficient detection and response. Machine learning, such as supervised and unsupervised learning, have been adopted to automate the detection of known threats as well as anomalies [8]. Furthermore, an automated response with pre-defined actions is also employed upon detection [10]. However, these ML techniques usually require a large amount of dataset for training which relies on a set number of features for the detection model and the scripted autonomous response is not able to effectively cope with the benign traffic [10].

ACD using RL can allow the implementation of decision-making agents that can take appropriate actions against advanced threats [10]. As stated earlier, ACD can be implemented with hard-coded rules for agents to make decisions upon threat detection; however, RL can augment it by allowing agents to learn their behaviors and adapt to changing threats and the environment. In addition, agents learn the consequences of their own actions and make decisions on how to respond to threats in real time. Moreover, decision making in RL is dynamic and the learning is primarily driven by rewards or penalties that an agent receives, therefore, labeled or unlabeled datasets to train are not a requirement. Furthermore, by introducing multiple agents, RL in ACD can be applied on a larger scale, where multiple agents can take coordinated actions for protecting the system.

Game theoretic approaches using MARL have seen a lot of success in recent years within various applications such as traffic control at intersections and robot path planning [11]. However, very few studies have explored the potential of applying MARL-based algorithms to cyber defence on an enterprise network [8]. The ones that have applied MARL in the cybersecurity of computer networks are mostly limited to intrusion detection systems. In fact, a model for multi-agent based IDS was proposed by Bhosale et al. [12]. Their system includes three agents on either the same or separate hosts and the agents are trained independently using information fed from a local database, with very little to no coordination between the agents. A recent study conducted by Wiebe et al. is the most relevant research in applying MARL to a simulated cyber environment

[13]. The authors explore the applicability of cooperative MARL in various cyber defence scenarios and demonstrate MARL's capability to develop effective cyber defence strategies.

## 1.2 Statement of Deficiency

Although recent research demonstrated that cooperative MARL is applicable for ACD, the methods that were applied neither showed any communication between the agents nor any coordination of actions [13]. The lack of coordination highlights an important area for improvement and necessitates a deeper understanding and interpretation of cooperative mechanisms formulated by the MARL agents. Agents coordinating their actions is like the process of teams of cybersecurity analysts that are domain experts, that work together and typically coordinate their actions to perform incidence response and mitigation. Therefore, for MARL to closely resemble the collaborative efforts of expert cybersecurity teams, the behaviour of the agents needs to be better explained with algorithms enabling them to do so.

Moreover, many real-life problems are only partially observable which necessitates communication of sharing experiences and observations. This is important as without communication, the defender (blue) agents may miss vital information regarding the attacker's (red) activities in another agent's field of view. They may fail to coordinate their defence effectively allowing the red agent to compromise hosts across subnets. By relaying key information such as the location of the attacker or an imminent threat, the blue agents collectively can isolate the red as well as learn from each other's behaviors. Additionally, there is a significant gap in applying MARL to cybersecurity, specifically in the emerging field of ACD. Therefore, this research intends to utilize MARL and introduce inter-agent communication to further this research space.

## 1.3 Aim

The aim of this research is to demonstrate the importance of communication between blue agents by showing that relaying key information will allow these agents to stop a malicious actor from compromising hosts across subnets.

This research seeks to expand upon the study conducted by Wiebe et al., by developing autonomous agents with a deliberate communication strategy for coordination [13]. To accomplish this, agents are trained using a communication enabled MARL algorithm, that allows each agent to not only take an environment action but also a communication action at every time-step. The communication protocol is not predefined but is learned along with the policies for the environment actions. Furthermore, Cyber-MARL (CyMARL) environment developed by the authors in [13] is leveraged in this research, and allows multiple agents to learn defensive tactics in a cyber defence scenario. The trained agents are then evaluated and assessed based on the following:

1. A quantitative measurement with the average reward as a key metric for comparing the results with a baseline.

2. An analysis of the content, context and timing of the learned communication.

The tactical strategies and behaviors learned by the communicating agents can contribute to the development of more advanced defender agents that can be tested in emulator environments before deployment on real networks. To achieve this goal, further research in this field, especially in the development of network simulators with more realism is warranted.

## 1.4 Research Activities

The following activities are conducted to demonstrate the importance of communication between agents:

1. Establish a benchmark cooperative MARL system and environment:

   CyMARL environment incorporates a MARL library (PyMARL2) [14], and Cyber Operations Research Gym (CybORG) for training and evaluating defender agents in cooperative MARL [15]. Several scenarios were established by Wiebe et al. in this environment and from these, a single confidentiality scenario is adapted and configured to formulate a baseline. This baseline allowed defender agents to learn tactical-level decision-making to stop the malicious actor by utilizing host-based monitoring data.

2. Implement a communication enabled MARL algorithm:

   Different approaches in the learning to communicate paradigm are investigated. These approaches are mapped to the problem of ACD. By conducting an in-depth analysis and by varying their conditions, the feasibility of different communication techniques in the simulated scenario are evaluated. A learning to communicate MARL algorithm is selected and implemented in CyMARL.

3. Investigate communication strategy and define the game design:

   Preliminary experiments are conducted where factors influencing the communication strategy are explored. The content and the timing of the communication is analysed which laid the foundation of the appropriate game design for further experiments. Based on the findings from the preliminary experiments, a scenario is designed which allowed the defender agents to have meaningful communication, that led to a coordinated effect in stopping the adversary from lateral movement.

4. Expand complex game design elements:

   A 'block' action is implemented which further augmented the defence capabilities of the defender agents. As an important component of a comprehensive cyber defence strategy, this block action allows the blue agents to stop all traffic from a specific zone. Different experiments are conducted in the study of this proactive measure to stop potentially harmful traffic or activities from reaching their intended target. Furthermore, to test scalability, more complex game design elements are pursued. Different scenarios are devised to test the communication strategies in larger and more complex problem spaces.

5. Evaluate communication performance:

   The effectiveness of the communication design in the presence of diverse adversarial strategies and network architectures are assessed in comparison with the non-communicative strategies set in the baseline.

## 1.5 Contributions

The biggest contribution made by this study is the successful validation of inter-agent communication within ACD that applies multi-agent systems. The agents developed in this research not only autonomously responded to threats but also meaningfully communicate before making their decision. Furthermore, this research highlights some important limitations of multi-agent ACD simulators that challenges the ability of agents to collaboratively address cyber threats.

## 1.6 Organization

Chapter 2 will provide background information on RL, Deep Learning, and MARL, with a review of learning to communicate algorithms, and an analysis of their applicability in the ACD environments. Chapter 3 discusses related work where different ACD environments are assessed. A study of the state of RL in cybersecurity is also provided in this chapter. Chapter 4 describes the methodology that is adopted for this research. Chapter 5 presents the evaluation and results that demonstrate the importance of communication between defender agents in CybORG. Chapter 6 provides the conclusion and future work.

# 2 Background

This research combines two areas of study: multi agent reinforcement learning and cyber defence. Within MARL, learning to communicate is an emerging paradigm where agents discover a communication protocol in order to make informed decisions in an environment. This emerging trend is the focus of this research wherein the learning to communicate paradigm will be applied to autonomous cyber defence. This chapter details the core concepts of RL, Deep learning and MARL that are relevant to this thesis and leads into the learning to communicate paradigm. RL will be discussed first with a brief description of the tabular methods. Deep learning will be covered next with an emphasis on recurrent neural networks. Additionally, an overview of MARL will be outlined, with different approaches to training the agents, and benefits and challenges of training multiple agents. Lastly, the theory of communication between agents will be examined with a detailed overview of state of the art algorithms.

## 2.1 Reinforcement Learning

Reinforcement Learning is a branch in ML that is distinct from other ML streams where an agent is enabled to make a sequence of decisions to achieve a goal. Unlike supervised learning, where a model learns from a labelled dataset for classification tasks, and unsupervised learning, which seeks to find patterns or structures in data without labels, prior knowledge in RL is not a requirement. Learning in RL is primarily driven by reward signals where an agent receives a reward or a penalty for the decisions it makes in the environment it interacts with. Decision-making policies are learned via an iterative trial and error process where at every timestep $t$, an agent has a pool of actions available to choose from. Moreover, rewards received by an agent are not necessarily instantaneous, and therefore the sequence of decision-making unfolds over several timesteps. The underlying goal of this agent is to maximize the cumulative scalar reward over time [4]. It can be seen in Figure 2.1, that when an agent chooses an action $A_t$, it observes a reward $R_{t+1}$ and a new state $S_{t+1}$.

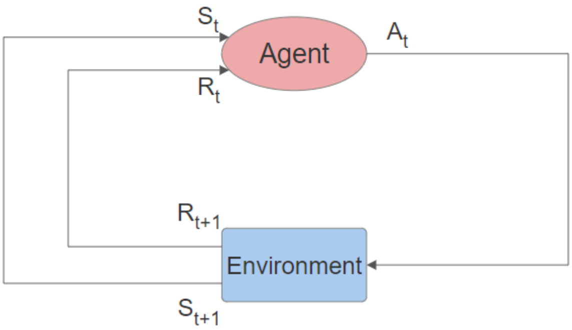


**Figure 2.1:** A single agent interacting with an environment.

### 2.1.1 Markov Decision Process

Decision-making in RL is typically modelled as a Markov Decision Process (MDP) which is an extension of Markov processes (or Markov chains). A Markov process is described as a sequence of events in which the probability of state transition to the next state $S_{t+1}$ only depends on the current state $S_t$ and so the history that led to the current state can be discarded. A Markov process consists of a tuple ($S$, $P$), where $S$ is a set of states, and $P$ is the probability of the state transitioning from $S_t \rightarrow S_{t+1}$ [4]. Equations 2.1 and 2.2 below express the Markov property and the state transition function.

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t] \tag{2.1}$$

$$P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s] \tag{2.2}$$

Building upon this foundation, a MDP introduces three additional components to the Markov process tuple ($S$, $A$, $P$, $R$, $\gamma$), where $A$ is the set of actions, $R$ is the reward function for the state transition and $\gamma \in [0, 1]$ is a constant that discounts future rewards to prioritize immediate or future returns. The transition to a new state not only depends on the current state, but also on the action taken and the reward received is a function of the state transition and action as described in 2.3 and 2.4 [4]:

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a] \tag{2.3}$$

$$R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \tag{2.4}$$

The notion of return in MDP encompasses the cumulative future rewards, that emphasises the goal of maximizing long term return rather than immediate rewards. This return $G_t$ can be written as a sum of all discounted future rewards in the following equation 2.5:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.5}$$

A policy $\pi$ in this context is a mapping that defines the probability of choosing an action given the current state, that guides the agent's decision-making process in 2.6:

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s] \tag{2.6}$$

The equations above can now be used to formalise the Bellman expectation equations for state-value functions and action-value functions. These equations express the recursive relationship between the value of a current state (or a state-action pair) and the values of the next states. The state-value function $v_\pi(s)$ is defined as the expected return starting in state $s$ and by following a policy $\pi$.

$$\begin{aligned} v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_{a \in A} \pi(a|s)(R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s')) \end{aligned} \tag{2.7}$$

The action-value function $q_\pi(s,a)$ is defined as the expected return starting in state $s$, taking the action $a$ and by following a policy $\pi$ [4].

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \\ &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a \in A} \pi(a'|s') q_\pi(s', a') \end{aligned} \tag{2.8}$$

The Bellman expectation equations above evaluate how good a given policy is but does not reveal the best way to behave in a MDP. In order to solve a RL problem, a policy needs to be derived that maximises the return over time. The optimal state-value and action-value functions are the ones that generate the maximum values in comparison to other values. To find these, the equations above are refined into the Bellman optimality equations as follows [4]:

$$v^*(s) = \max_a [R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s')] \tag{2.9}$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q^*(s', a') \tag{2.10}$$

Most RL problems can be solved by leveraging these Bellman equations in some capacity that iteratively updates the value functions and improves the policy towards optimality. These principles make the framework of MDP versatile, enabling it to solve a wide range of problems such as robotics and even tactical level decision making in cybersecurity.

### 2.1.2 Classification of RL algorithms

A RL algorithm can be defined as a method for training agents to make decisions in an environment to achieve a goal [4]. Through RL, an agent learns the mapping of states to actions (deterministic) or states to action probabilities (stochastic). RL algorithms are broadly categorized based on the model of the environment they use, if they are value-based or policy-based and the type of policy they follow.

**Model-Free vs Model-Based**: Model-free algorithms train agents directly from the experiences of the interaction with the environment. Meaning, agents learn a policy or a value function based on the states and rewards observed and rely on trial and error in gradually refining the decision-making strategies. On the other hand, in model-based learning, a model of the environment is constructed, including the transition probabilities $P_{ss'}^a$ and the reward function $R_s^a$, for planning and decision-making. This allows the agent to simulate the environment without directly interacting with it. In other words, agents can predict the next state and the reward without actually having to take an action in an environment [4].

**Value-Based vs Policy-Based**: Value-based learning is concerned with learning the value functions that generate an optimal policy. Note that the optimal policy is not generated directly but implicitly. These value functions provide a measure of how good it is to take a particular action in a certain state. In contrast, the policy optimization methods also known as policy-based methods, directly optimize the policies that maximize the expected return as shown in the policy gradient Equation 2.11 below. These methods seek to adjust the parameters of the policy via gradient ascent on the expected return. An obvious benefit of directly handling a policy is that it allows the agents to learn a continuous action space and also enables them to learn stochastic policies [7].

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[ \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) R_t \right], \tag{2.11}$$

$J(\theta)$ represents an objective function and $\theta$ represents the policy parameter.

**Off-Policy vs On-Policy**: Off-policy algorithms can learn the target policy from data generated from following a different policy. This approach allows learning to be flexible where an agent is able to learn from sampled historical data (experience replay) or from exploring multiple strategies simultaneously. On-Policy algorithms are the methods that learn policies by directly following and updating them. Meaning, an agent takes actions in the environment according to its current policy and then updates the policy based on the action taken. The policy being improved is the same one used to interact with the environment [4].

### 2.1.3 Q-Learning

Q-Learning is a model-free, value-based, off-policy algorithm in RL, that seeks to find an optimal policy for any given finite MDP. It operates by learning an action-value function that ultimately allows an agent to take an action in a given state. The algorithm is one of the earliest tabular forms of learning, where q-values for each state-action pair are stored in a table [4].

As it can be seen in Algorithm 1, the Q function is updated iteratively using the Bellman equation, as the agent explores in an environment [4]. This tabular approach is obviously limited for RL problems with large and continuous state spaces. To address these issues, Q-Learning and other RL algorithms adopted function approximation. Techniques such as linear regression were used to approximate the Q-values instead of storing them in tables. This approach allows for generalization across similar states but were still limited in handling more complex environments. Nonetheless, these techniques provided a foundation for future algorithms that leverage deep neural networks which will be discussed in the next section.

A few notable terms introduced in the Q-Learning Algorithm are as follows:

- Episode: This refers to a trajectory of an agent from an initial state to a terminal state, through an environment. An episode usually ends when an agent reaches the final state but also can be a fixed number of timesteps. The game is reset after the end of an episode and repeated for agents to learn an optimal policy.

**Algorithm 1** Q-Learning Algorithm for estimating $\pi \approx \pi_*$

```
Algorithm params: step size $\alpha \in$(0,1] , small $\epsilon > 0$
Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for each episode do
    Initialize state $s$
    for each step of episode do
        Choose action $a$ from state $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $a$, observe reward $r$ and next state $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
    end for
end for
```

- $\epsilon$: This refers to the exploration rate of an agent. Depending on this rate, it allows the agent to sometimes explore new actions in a given state and at other times follow the learned policy.
- $\alpha$: This represents the learning rate, which scales the size of the updates of the value function.

## 2.2 Deep Learning

Deep Learning (DL) is another subset of ML, that utilizes Artificial Neural Networks (ANN) in order to learn representations directly from data. ANN's are composed of nodes (neurons) which are interconnected through an input layer to the output layer, and includes one or more hidden layers. The distinction between a Deep Neural Network (DNN) and a shallow network lies in the fact that there are multiple hidden layers within a DNN.

As depicted in Figure 2.2, data in a neural network is fed to the input layer, which is then passed to the neurons in the hidden layers through a system of weighted connections. A weight, essentially, is the importance of a particular neuron. Basically, the values of the input layer are multiplied with the weights and the sum of all weighted inputs is then sent to the nodes in the hidden layers. This weighted sum goes through an activation function which subsequently determines the output signal of a particular neuron. Additionally, each neuron within the hidden layers has a scalar value associated with it known as bias. Bias allows the model to shift the activation function towards left or right to better fit the data. In general, this whole process is sequentially applied across all hidden layers until reaching the output layer, which outputs the node with the highest value representing the model's prediction [16]. However, there are other architectural possibilities depending on the type of neural network and the specific task such as convolutional and pooling layers in Convolutional Neural Networks (CNNs), recurrent and attention mechanisms in Recurrent Neural Networks (RNNs) and transformers respectively [16].

The training of a neural network involves backpropagation which is virtually

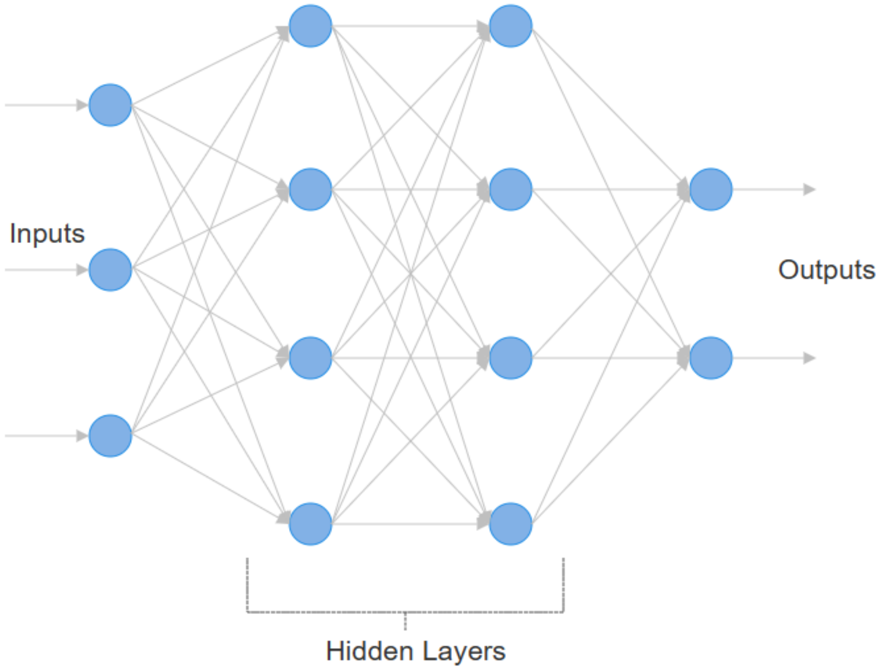


**Figure 2.2:** An example of a neural network with two hidden layers.

the reverse of the forward propagation process described above. During backpropagation, the information flows from the output layer to the hidden layers. Here, the model uses a loss function to evaluate its accuracy on the predicted output value by calculating the deviation (error) of the predicted value from the actual output. To minimize this error, the model typically employs gradient descent which iteratively adjusts the weights and biases by calculating the gradient of the loss function with respect to each weight and bias [16].

An important aspect of deep learning is that the model can learn by processing large amounts of raw data without a need for manual feature manipulation and extraction which is usually the case with supervised and unsupervised learning. This has led to major advancements particularly in fields like image processing, speech recognition, and language translation where traditional algorithms had previously fallen short [7].

### 2.2.1 Deep Q-Networks

The application of DNN to RL, also known as Deep Reinforcement Learning (DRL), posed a significant challenge initially. The issue arises in the fact that deep learning models learn by using large samples of datasets to make their predictions. However, in RL, data is generated sequentially as agents take their actions and receive their rewards which proved infeasible in learning DNNs. To combat these challenges, Mnih et al. proposed Deep Q-Networks (DQN) [17].

In DQN, outlined in Algorithm 2, an online network (Q-network) actively learns and approximates the Q-value function during training. Thus, it is respon-

**Algorithm 2** Deep Q-Network (DQN) Algorithm

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, M$ do
    Observe initial state $s_1$
    for t $= 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q(s_t, a; \theta)$
        Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample random mini-batch of transitions $(s, a, r, s')$ from $D$
        Set $y = r$ if episode terminates at step $t + 1$
        otherwise set $y = r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-)$
        Perform a gradient descent step on $(y - Q(s, a; \theta))^2$ with respect to the network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    end for
end for
```

sible to make predictions of agent actions based on the current state. The key innovation of this algorithm is the use of experience replay, which provides enough samples, and an additional neural network (target network), which stabilises the training of the main network. Experience replay stores agent experiences (actions in a given state, rewards etc.) in a replay buffer and after enough samples have been collected, the buffer is then sampled again to train the main network. The target network has the same architecture as the main network but its weights are updated less frequently. The use of the secondary network, prevents the moving target problem, where the Q-value updates of the main network are not as significant, and are aligned with the target network [17]. This integration of deep learning with Q-Learning was a breakthrough that significantly improved the ability of RL algorithms to solve problems in large state spaces, such as atari games that provide pixel input as observations.

### 2.2.2 Recurrent Neural Networks

The most basic type of deep neural networks are Multi-Layer Perceptrons (MLP) which are fully connected feed forward networks. Typically, these networks take a set number of inputs and have fixed-size outputs. And although, these models are powerful in solving many complex problems, they have limitations especially with processing sequences of data.

Unlike MLP, RNN can have variable input length and have connections that loop within the hidden layers. This means that RNN use not only the current input for their prediction, but also the hidden state from the previous input, which allows the knowledge from the previous input to persist. This memory characteristic makes RNN particularly suited for tasks that involve sequential inputs, such as time series analysis [16].

### 2.2.3 Deep Recurrent Q-Networks

While MDP scenarios require the visibility of the entire state, there exists another framework called Partially Observable Markov Decision Process (POMDP) that enable agents in RL to solve problems without complete access to an environment. This obviously adds a layer to the complexity where agents now must make decisions based on partial observations. The DQN algorithm, discussed in Section 2.2.1, is designed for situations where an environment is fully observable and inherently assumes that the observations provided are a complete representation of the environment state, and therefore struggles in a POMDP framework.

To address this deficiency of DQN in POMDP, Hausknecht and Stone proposed Deep Recurrent Q-Networks (DRQN), an extension of DQN algorithm that incorporates RNN, specifically the Long Short-Term Memory (LSTM) network, into the DQN architecture [18]. With this integration, past observations became accessible via the hidden states due to the memory mechanism of RNN. Along with the observations from the current timestep, DRQN agents are enabled to aggregate observations over time which allows agent's to make more informed decisions in a partially observable environment.

## 2.3 Multi Agent Reinforcement Learning

Many real-world problems cannot be solved by a single agent system, and therefore multi-agent systems are necessary. MARL extends RL by allowing multiple agents to interact with one another and the environment. The main difference is that, in MARL, the environment state is represented as a joint state that includes individual observations of all agents; the state transitions are then the result of joint actions of all agents. Rewards for each agent are based on a combination of the joint state and joint actions [19]. Figure 2.3 shows multiple agents interacting with the environment.

Cooperative MARL as noted in Chapter 1, is a research area where multiple agents cooperate and potentially coordinate their actions to achieve a common goal. Cooperative algorithms are generally classified by the way agents are trained. The most common approaches to training within MARL are: independent learning, and centralized learning with decentralized execution.

**Independent Learning**: In this setting, each agent learns its policy for their own actions by observing a state and the agents receive a joint reward as seen in Figure 2.4a [19]. The learned policies are naturally executed independently as well. This is sometimes also known as decentralized training and execution paradigm. Experiences and observations of other agents may be monitored but it does not play any part in making decisions. An example of this approach is the independent DQN, an extension of single agent DQN, where each agent learns its own Q-value function independently, without explicitly considering the policies or behaviors of other agents [20].

**Centralized Learning with Decentralized Execution (CLDE)**: This paradigm has been extensively studied in recent years and has proven to be better

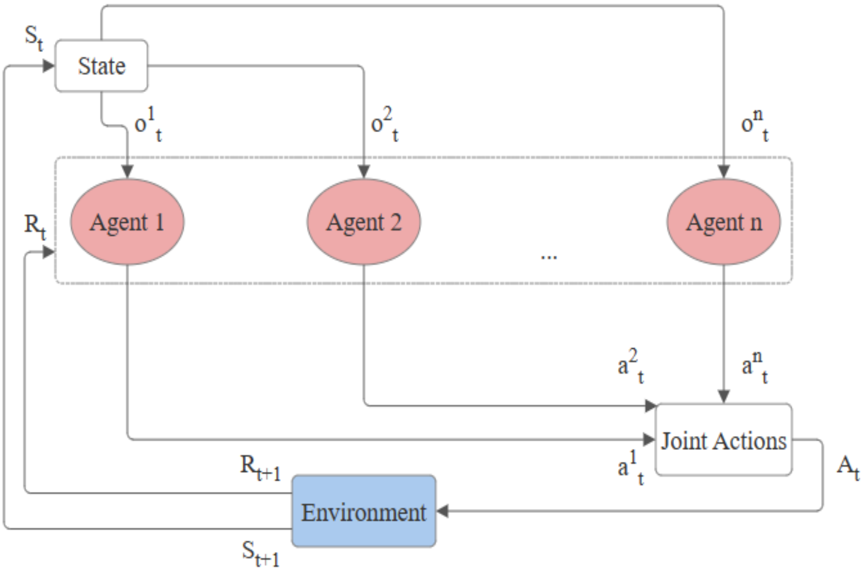


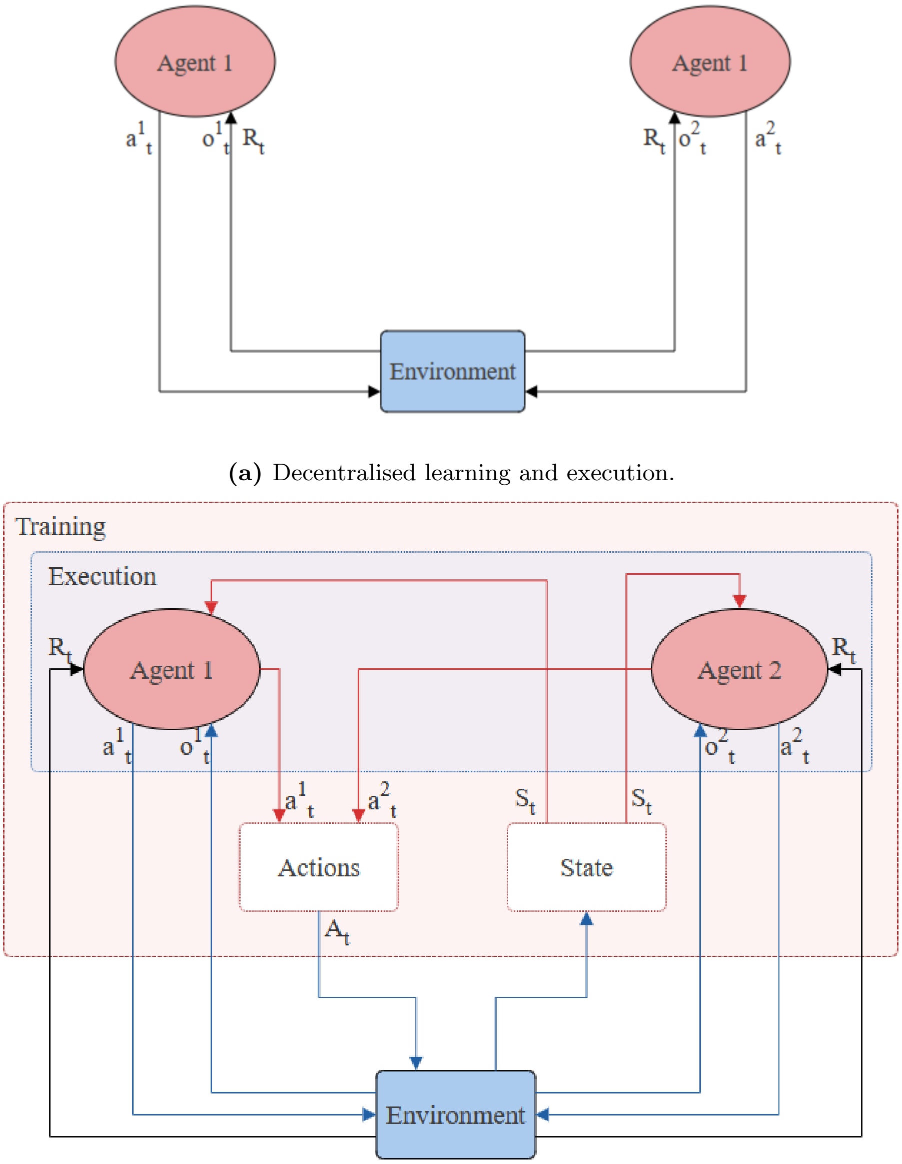
**Figure 2.3:** Multiple agents interacting with the environment.

performing over fully decentralized learning algorithms. It differs from other frameworks in the way agents coordinate during training versus during execution. Depending on the algorithm, in the training phase, each agent is allowed to observe policies or states of all other agents and usually there is no limit to this coordination [19] as seen in Figure 2.4b. However, during execution, the agents act independently on the learned policies with limited to no communication. This technique provides an obvious benefit of reduced overhead during execution. One such example of CLDE is QMix, which is a popular Q-Learning algorithm for cooperative MARL and discussed further in Section 2.3.2.

### 2.3.1 Advantages and Challenges in MARL

One of the main advantages of MARL algorithms is their ability to decompose a larger problem space of single agent RL into smaller areas of responsibility. It also offers several other advantages such as robustness and scalability over single agent RL. If an agent is compromised in a system, others can take over the responsibilities of the failed agent. Moreover, if more agents are needed to accomplish any given task, new agents can be introduced. MARL also provides benefits of experience sharing. This allows for a faster training process and results in more efficient use of the resources. However, introducing multiple agents also brings forward many challenges which can make it difficult for agents to find an optimal policy [21].

Single agent algorithms, like the tabular Q-Learning, calculate all state-action values known as Q-values for every possible state or state-action pair, which becomes computationally infeasible as the number of dimensions grow. In MARL, this is compounded as the joint state-action space increases exponentially with the number of agents. This is known as the curse of dimensionality. Moreover,



**(a)** Decentralised learning and execution.

**(b)** Centralised learning and execution.

**Figure 2.4:** Decentralised Learning vs CLDE.

agents in MARL learn concurrently and therefore the best policy of each agent changes while other agents take their actions causing the environment to become non-stationary. Additionally, many real-world problems are only partially observable which necessitates coordination and communication of sharing experiences and observations. However, it is especially challenging to design algorithms that can impart messages to other agents. Sharing of actions taken, observations and how much to share over limited bandwidth all depend on the efficiency of the designed algorithm. A lot of recent work has primarily focused on dealing with these challenges such as combining deep neural networks with Multi Agent

Reinforcement Learning, adopting varying learning rates and designing communication mechanisms and protocols [21].

### 2.3.2 QMix

Decentralized training and execution such as independent DQN have performed well for relatively simple tasks such as the atari video games. But for more complex tasks such as the Star Craft II challenge, they fail to converge as it renders the environment non-stationary. Several methods for centralized training have drawbacks such as requiring prior knowledge, for example, coordination graphs and sparse cooperative Q-learning [22]. Even for CLDE, some methods are impractical in scaling the number of agents and restricted to on-policy learning only as seen in Counterfactual Multi-Agent (COMA) [23]. Sometimes they only represent simple action-value functions and cannot learn extra state information during training as observed in Value Decomposition Networks (VDN) [24]. Therefore QMix was introduced in [22] within the CLDE paradigm, which improves upon all the drawbacks mentioned above.

During learning, QMix architecture incorporates a single feed-forward mixing network along with DRQN networks (agent network) aseen in Figure 2.5. Independent agent policies (Q-values) are output from all agent networks which are then passed to the mixing network. This mixing neural network aggregates all individual Q-values of each agent into a global Q-value and is constrained to ensure that the joint action-value function is monotonic with respect to each agents individual Q-value. This constraint aligns the agent's objectives and guarantees that if an agent's individual Q-value changes while others remain fixed, the overall Q-value cannot decrease. Additionally, the parameters of the mixing network are generated by a separate hyper-network, which takes the global state of the environment as an input. Subsequently, the $Q_{tot}$ output from the mixing network is then decomposed into individual values which are used for agent actions in the environment. It is important to note that, during execution, mixing network plays no part in the decision-making and the agents act independently based on the Q-values generated by the agent networks [22].

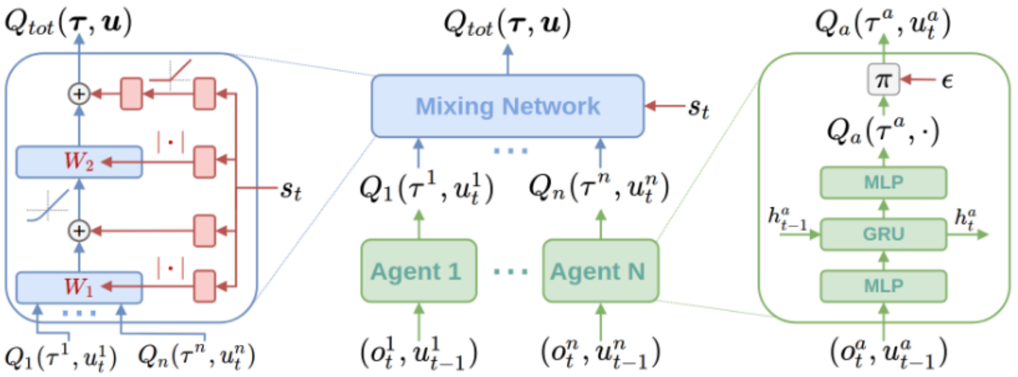


**Figure 2.5:** QMix architecture [22].

## 2.4 Learning to Communicate

While CLDE paradigm, without any coordination during execution have been successfully applied in many applications, it has drawbacks. One major issue is it limits the ability of agents to respond to situations that require coordinated actions during execution. It is also limiting when agents do not have a global view of the state, especially if they are not properly addressed during training. Centralised execution may be beneficial in overcoming the above shortcomings, but it suffers with scalability and information overhead. For example, for cyber defence in an enterprise network, bandwidth may be limited and dealing with the vast amount of network traffic becomes impractical.

This leads us to the *learning to communicate* paradigm. This technique allows agents to not only send messages during learning but also during execution. This area has been extensively researched in the last few years, and has led to many solutions to enable agents to coordinate by communicating. While some approaches use CLDE and enable agents to fully cooperate during training while allowing them to send limited bandwidth messages during executing [25]. Others have applied the learning to communicate paradigm fully centralised [26] as well as decentralised policies [27]. Below is a review of some of the state of the art algorithms in this paradigm. They all have their advantages and disadvantages, listed in Table 2.1, where algorithms may be only suited to very specific tasks that they were designed for.

### 2.4.1 RIAL and DIAL

Foerster et al. proposed two methods, namely **Reinforced Inter-Agent Learning (RIAL)** and **Differentiable Inter-Agent Learning (DIAL)** for communication for fully cooperative and partially observable problems [25]. Both approaches are based on DRQN that employ the CLDE framework. In this setting, the agents can fully communicate without any restrictions during learning, but during execution, communication is restricted via the limited-bandwidth channel. As can be deduced, the application of CLDE is atypical in these algorithms, in which they allow for some abstract information to be conveyed directly even after the policies have been trained.

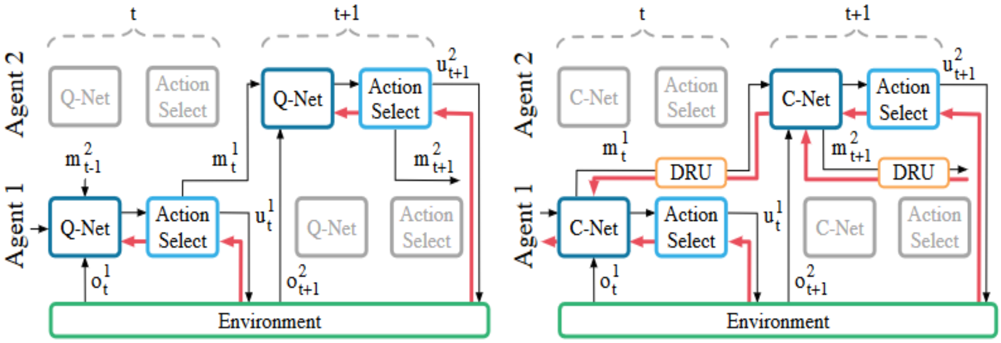
In RIAL, the Q-network is separate for all agents where the network for each agent is split in two, one for the environment actions and the other for the communication actions. The action selector then determines the appropriate actions for communication and the environment. It is important to note that the Q-Networks, that includes the q-values for both, environment and messages are trained based on the independent DQN foundation and does not take advantage of centralized learning. Therefore, once a message is sent by an agent, the feedback of the received message is not provided to the sender. Authors also experimented with training a singular network for all agents which is known as parameter sharing. They showed that RIAL benefited from centralized training with this approach where the agents learn a common network but are still able to deduce a specialised policy based on the inputs the agent provides [25]. The author's intuition was that in a real-world interaction between two people, the listener

| Algorithm | Features | Advantages | Disadvantages |
|---|---|---|---|
| **RIAL** | DRQN with CLDE.<br>Parameter sharing. | Interpretable communication. | Lacks feedback mechanism. |
| **DIAL** | DRQN with CLDE.<br>Gradients for communication.<br>Discrete communication. | Feedback between agents during learning.<br>Interpretable communication. | Computationally expensive when scaled. |
| **CommNet** | Average hidden states of other agents as inputs.<br>Continuous communication | Integration with standard RL algorithms. | Communication hidden within neural network.<br>Central Controller. |
| **BiCNet** | Multi-agent actor-critic framework.<br>Policy network generates actions based on shared observations and local views.<br>Q-network evaluates the actions taken by agents. | Scalability due to parameter sharing. | Increased computational requirements.<br>Communication hidden within neural network.<br>Central Controller. |
| **TarMAC** | Targeted communication with learned message addressing.<br>Signature-based soft attention mechanism. | Adaptive to variable team sizes and diverse environments. | Multi-round communication approach. |
| **IC3Net** | Gating mechanism to control continuous communication.<br>Individualized rewards for each agent. | Designed for heterogenous agents.<br>Applicable to cooperative and competitive settings. | Optimal communication in large-scale environments may be challenging. |

**Table 2.1:** Comparison of RIAL, DIAL, CommNet, BiCNet, TarMAC, and IC3Net algorithms

must provide some indicators of the level of understanding of a conversation to the speaker. Based on this principle and on the fact that RIAL is limited in the fact that agents are unable to provide feedback on each other's communication actions, DIAL was introduced.

In DIAL, during centralized learning, both the communication and the environment actions are output from a C-Net with a direct connection between one agent's network output and another agent's network input as seen in Figure 2.6. This allows agents to send direct and continuous real-valued messages to each other that uses backpropagation of the gradient via the communication channel. The feedback mechanism functions such that when an agent acts based on the received communication, the resultant reward informs the value of the communication where positive outcomes reinforce the message and negative results deter



(a) RIAL - RL based communication

(b) DIAL - Differentiable communication

**Figure 2.6:** Communication flow in RIAL and DIAL [25].

its future use. During execution, learned messages are transformed into a discrete set of communication actions suitable for the task.

The C-Net for DIAL as illustrated in Figure 2.7 is a RNN which has two hidden layers $h$ that are interconnected and maintained for the entirety of an episode. The architecture is designed such that it takes four inputs as a tuple $(o_t^a,\ m_{t-1}^{a'},\ u_{t-1}^a,\ a)$ that produces $z_t^a$ at each timestep. Subsequently, $z_t^a$ is processed through a 2-layer RNN alongside the first hidden layer $h_{1,t}^a$ from the previous timestep to produce the outputs. $a$ is the agent identifier, input $o_t^a$ is the partial observation of agent $a$ at time $t$, $m_{t-1}^{a'}$ is the message received from agent $a'$ from previous timestep and $u_{t-1}^a$ is previous action of agent $a$. The output at timestep $t$ consists of $q$ values for the environment actions that are fed to the action selector and $q$ values for the message that are transmitted to a Discretize/Regularize unit (DRU). During learning, DRU regularizes the communication signal using a sigmoid function and during execution, it converts the signal into a discrete binary value. While, DIAL can handle continuous messages as they are part of the centralized training process, the authors choose to discretize the real-valued messages as they aligned better with the experiments they conducted, for instance, representing the light bulb switch as a binary communication mechanism. Also, to note, DIAL allows to increase from 1-bit to a larger message space depending on the task [25]. The algorithm for DIAL can be found in Appendix A.

The authors of DIAL experimented with two simple tasks that are partially observable and require cooperation and coordination from agents [25]. The first activity is the prisoner switch riddle, where one hundred prisoners are isolated and each one is randomly and daily brought into a room with a light bulb. The task for the prisoners is to announce if everyone has visited the room. If they announce it correctly, they are all set free but should they be wrong then everyone will be executed. The authors formalised this game with three and four prisoners where each prisoner is represented as an agent and the light bulb serves as a communication channel through which agents can send a message to the next agent in the room.

The next task involved two agents receiving a random MNIST digit $\in$ [0,...,9] and each agent is to determine the other agent's digit. The game was designed

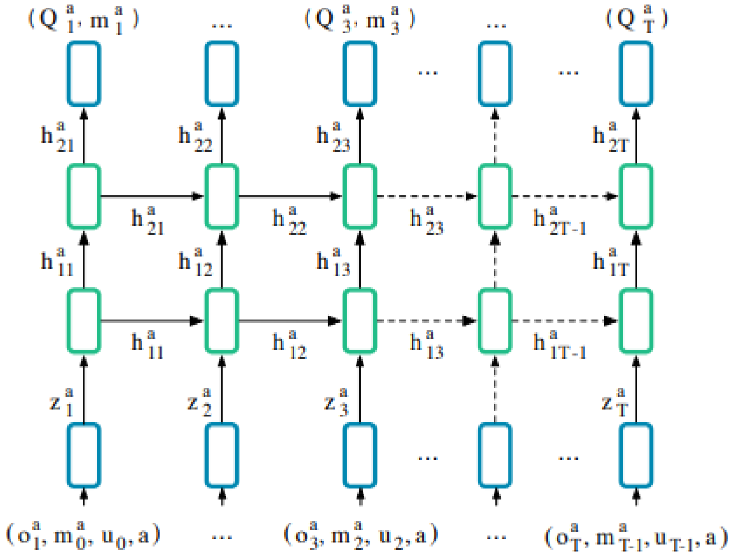


**Figure 2.7:** DIAL architecture [25].

such that each agent has four communication steps, followed by an environment action to derive the other agent's number. Experimental results on both of these tasks showed the success of DIAL in learning effective communication protocols. Agents in each case derived an encoding that unfolded over the number of timesteps allowed, and solved these games.

### 2.4.2 CommNet

**Communication Neural Network (CommNet)** is another proposed algorithm allowing agents to communicate with each other before taking actions in multi-agent systems [26]. A continuous communication is realized with this model and the algorithm is trained via backpropagation. The model, as seen in Figure 2.8, involves a central controller that receives information about the state of all agents and their communication messages and uses multiple steps of communication to generate output actions for all agents. The process starts with encoding the state observations and setting the communication messages to zero. Then, in each round, the controller concatenates the previous hidden and communication states of all agents and passes them through a linear layer followed by a non-linear function to obtain new hidden and communication states. The final hidden state is decoded to provide a distribution over the action space.

The effectiveness of CommNet was tested on cooperative navigation tasks and compared with other methods in this field. The results showed that CommNet produced communication strategies that were easy to understand and useful. However, the algorithm assumes full cooperation between agents at every timestep, which may not be true in all situations.

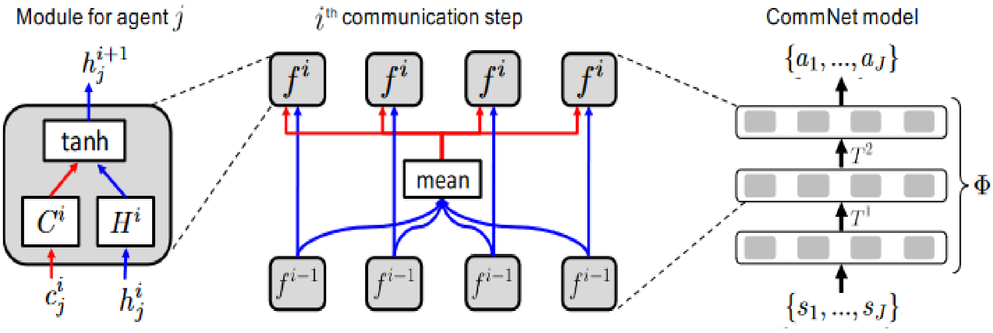
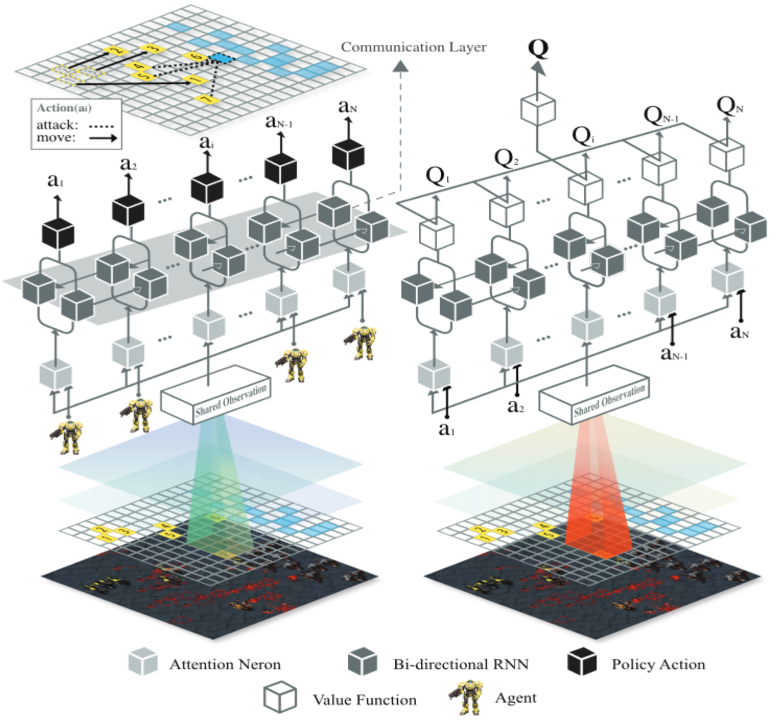


**Figure 2.8:** CommNet model [26].

### 2.4.3 BiCNet

**Bidirectionally-Coordinated Nets (BiCNet)** was proposed for training deep reinforcement learning agents with different parameters to communicate with each other using a RNN [28]. The architecture, depicted in Figure 2.9, comprises of a multi-agent actor (policy) and a multi-agent critic (Q-network) network. The policy network takes shared observations and local information to generate actions for all agents. The Bi-Directional recurrent network serves as a local memory, allowing individual agents to maintain their own internal states over



(a) Multiagent policy networks (b) Multiagent Q networks

**Figure 2.9:** BiCNet architecture [28].

time while sharing information with their neighbors. To compute the backward gradients, the network is unfolded for a certain number of time steps (N), which represents the length of the sequence of actions taken by the agents. Backpropagation Through Time (BPTT) is then applied to propagate the gradients back through the network, updating the parameters at each time step. The gradients are then aggregated from both the value function and the policy function, which determines the actions taken by the agents. This approach assumes that each agent is aware of the global state of the environment which may not hold true in a partially observable environment.

### 2.4.4 TarMAC

Another communication architecture called the **Targeted Multi Agent Communication (TarMAC)** was proposed, where agents learn what messages to send and to whom to address them in order to perform cooperative tasks in partially-observable environments [29] and follows the centralized learning with decentralized approach. Figure 2.10 illustrates the TarMAC architecture. Each agent's policy is implemented as a 1-layer Gated Recurrent Unit (GRU) and takes the local observation and messages received from other agents as input to update its hidden state. Each message contains a signature that includes the intended recipient and a value which is the actual message. Each receiving agent learns the messages and computes attention weights for every incoming message. The attention weights are then aggregated and used as input for the local actor along with the local observation. The system is then trained using a centralized critic model. The communication system enables multi-round communication to increase efficiency. The goal of the system is to maximize the team reward while executing joint actions in the environment.

### 2.4.5 IC3Net

**Individualized Controlled Continuous Communication Model (IC3Net)** is introduced by Singh et al. not only for cooperative multi-agent settings but also for competitive and mixed scenarios, where agents can learn what and when to communicate based on a given scenario [30]. Figure 2.11 outlines the architecture for IC3Net and can be seen that the observations of each agent are passed to a Long Short Term Memory (LSTM) module, which is a variant of RNN. The observations are then encoded and relayed to the communication-action module. A gating communication mechanism is employed and controlled by a gating function, which contains a soft-max layer for two actions (communicate or not). Subsequently, the communication vector for each agent is calculated as a linear combination of the hidden states of all other agents and weighted by their binary communication actions. The binary action is determined based on the output of the gating function at the current time-step. Both the action policy and the gating function are trained using the REINFORCE algorithm [4].

The authors demonstrate the effectiveness of IC3Net through experiments in three different environments, including StarCraft, and show that it outperforms other baselines such as IC3Net without communication and CommNet [26] in

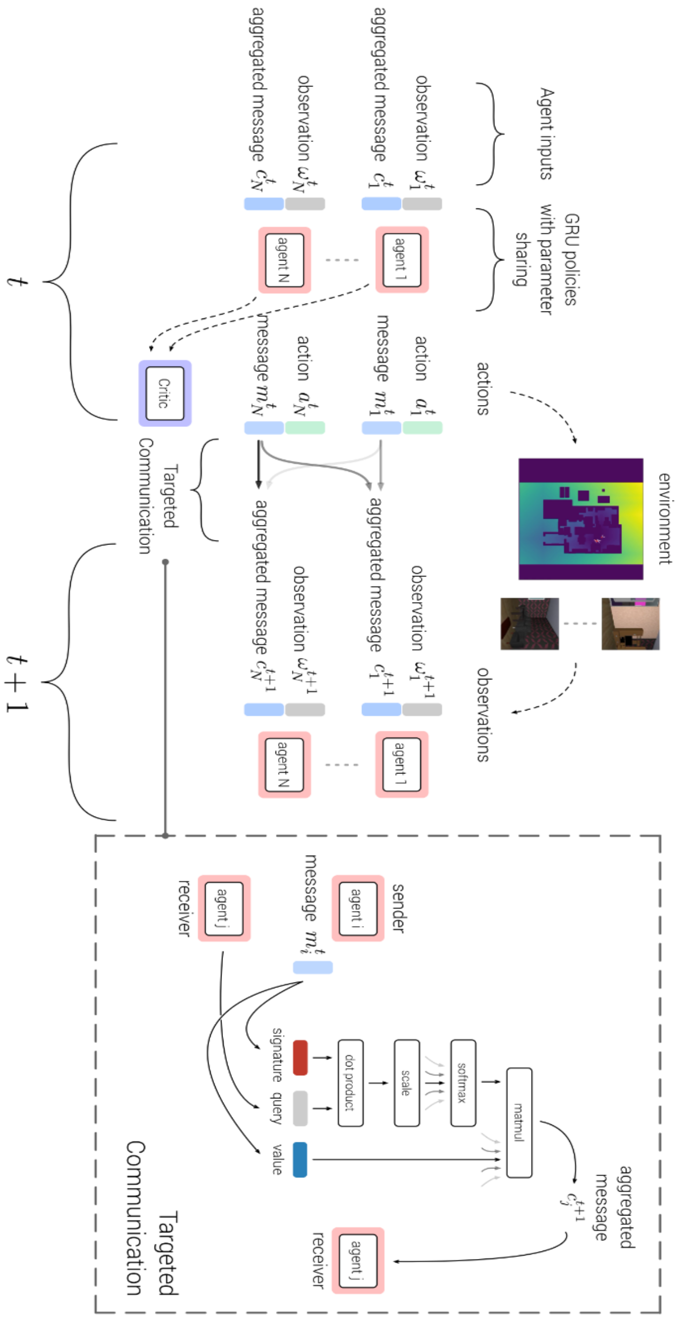


**Figure 2.10:** TarMAC architecture [29].

terms of convergence speed and final performance. However, the paper acknowledges some limitations of IC3Net, including the computational cost of training and the potential difficulty in scaling to very large agent populations. The authors suggest that future research could focus on addressing these limitations and further improving the performance of the model in more complex scenarios.

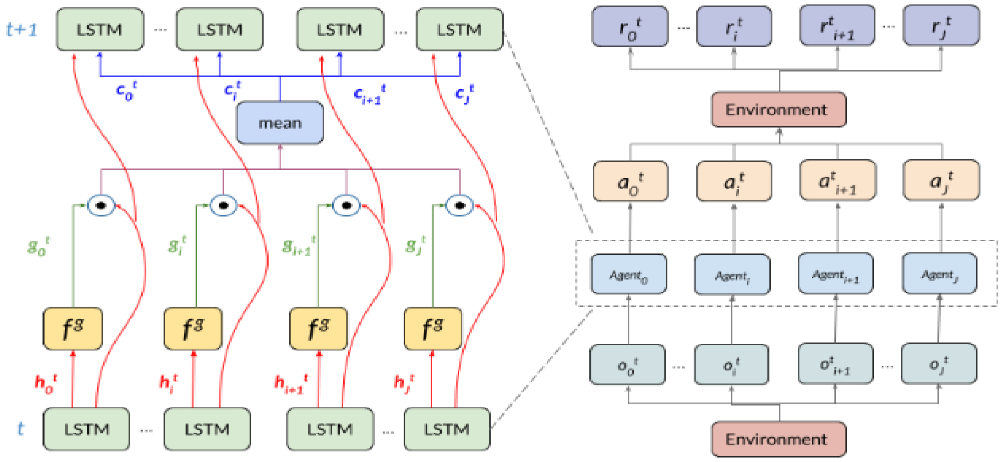


**Figure 2.11:** IC3Net architecture [30].

## 2.5 Summary

This chapter provides an overview of essential concepts and algorithms foundational to this thesis. It begins by discussing single-agent RL within the framework of MDPs and the Q-Learning algorithm. The chapter then transitions to DRL, highlighting the DQN algorithm and the role of RNNs in handling sequential data, leading to an exploration of DRQN for environments with partial observability. The chapter further delves into cooperative MARL, covering CLDE and the learning to communicate paradigm, where agents share information before taking actions. A particular focus is given to the DIAL algorithm, which extends DRQN by enabling direct agent-to-agent communication, thereby improving coordination and learning efficiency.

These advancements in MARL, especially the ability for agents to communicate, have significant implications for ACD. In such scenarios, multiple defensive agents can collaborate in real-time to detect and mitigate cyber threats more efficiently than independent agents. This chapter sets the foundation for the detailed exploration of the research presented in subsequent chapters, emphasizing the potential of MARL with communication for ACD systems.

# 3 Cybersecurity Related Work in Reinforcement Learning

In this chapter will present the current state of cybersecurity within the context of RL. Particularly, the composition of this chapter is divided such that the first section delves into the field of Autonomous Cyber Operations (ACO), with an overview of various ACO environments to train RL agents. Next, we will brief on various studies that have been conducted in the field of RL in ACO and lastly, we discuss some real world research of applying communication in MARL.

## 3.1 Autonomous Cyber Operations

So far, ACD has been introduced and discussed in Chapter 1, which is concerned with allowing blue agents to autonomously respond to threats in a network. However, Autonomous Cyber Operations (ACO) is a broader term that encompasses the autonomous decision making of both, the attacker as well as the defender agents in a network [15]. Training the red agents alongside blue is critical and serves two purposes: 1) It provides tools for the red team to automate the offensive, and 2) it enables the development of blue agents that can respond to dynamic threats rather than static attacks.

This research is focused on training blue agents against a scripted attacker agent with the argument that, to test the viability of any framework in autonomous defence, defender agents must be evaluated against a predefined adversary as a starting point. Naturally, the next step is then to validate these techniques against a well trained attacker, who can potentially mimic the threats of real-world settings, although this approach is left for future work.

RL, specifically DRL, has emerged as an important research area for ACO, that offers a potential for both, cyber defence as well as offensive capabilities for the red team. Unlike supervised and unsupervised learning, DRL can facilitate the simulation of red and blue agents in a network with a sequential gameplay where red and blue agents take turns, at every step, to deploy their actions. One of the key potentials of DRL in ACO is its ability to handle large state spaces, which is essential in environments where agents must make decisions based on vast amounts of data, such as the enterprise network traffic of real-time system activities [31]. Here DRL has the ability to extract meaningful patterns from network and host logs, thereby facilitating more strategic decision-making processes.

Moreover, DRL is proficient in managing large action spaces that is particularly useful in ACO where the agents must choose from a comprehensive list of possible actions [31]. These actions may include different mitigation techniques for blue agents or distinct exploit mechanisms that an attacker can employ at any given time. The dynamism of the red agent is particularly interesting, where blue agents can adapt their strategies based on the changing tactics of an attacker, ensuring that blue agents are well prepared for unpredictable threats. Additionally, MARL compliments the use of single agent RL, where now multiple agents can dissect the larger problem space with smaller areas of responsibility. For instance, although blue agents still have to learn to detect threats and their mitigation actions in their respective zones, they individually only have to process data on a smaller scale compared to one single agent monitoring the entire network.

However, developing agents using MARL on a real network is not feasible due to the nature of RL requiring many iterations in order to train an agent towards an optimal policy. Therefore, simulators and emulators must be used to design these agents. The broader idea is that once these agents are trained in a simulated network, they can then be validated on an emulated network (virtual machines) and subsequently can be deployed on a real network. The biggest challenge that is faced in this area of research is to replicate an actual network with realism and details in simulations.

Real-world networks are complex, where they are usually divided in segments for security reasons and also with a possibility of large amounts of benign network traffic [32]. Networks also consist of many hosts with different operating systems and unique vulnerabilities they may carry. To capture these intricacies of a real-world environment in a simulator is difficult and therefore must be simplified when transferring the details. Although simulations are cost-effective and eliminate risks associated with real networks or actual malware, it creates a gap where trained agents on a simulator may develop strategies that are ineffective beyond the simulated scenarios [33].

Emulated networks, on the other hand, offer a more realistic approach to training agents within the DRL and MARL frameworks. Unlike simulations, emulations replicate the characteristics of actual networks more closely, and provide a detailed representation of network behaviors. This allows for the development of agents that can interact with network dynamics that mirror real-world operations more accurately.

However, a significant drawback of using emulation is the substantial increase in training time. They require the setup and maintenance of a nearly full-scale system, which includes running actual network protocols and services. This not only demands more computational resources but also slows down the iteration speed of training cycles. While they bridge the realism gap than simpler simulations, the computational overhead and slower training processes pose challenges for scaling up experiments, especially when multiple agents are involved and need to interact in complex scenarios. For this particular reason, this research only utilizes a simulator and leaves the emulation aspect for future work.

### 3.1.1 ACO environments

Any RL research in any field requires an environment that allows the agents to make sequential decisions in that environment. Thus, many environments over the years have been developed for simulating a cyber space within the context of RL. Although the developed agents may not be ready to be used on a real network, yet they still provide a good foundation for all subsequent researches for autonomous agents.

Microsoft's **Cyber Battle Sim (CBS)** is one such simulator that is specifically implemented to train red agents that focus on moving laterally through a network [34]. The architecture of this environment includes a parameterized network where each node represents a windows host with specific vulnerabilities that an attacker can exploit to move through the network. The actions available to an agent include local attacks, remote attacks, and connecting to other nodes. This environment is only partially observable to the agent, where the attacker has to explore the network for full visibility. The environment is built on the Python-based OpenAI Gym interface, which can facilitate the training of agents using various RL algorithms. Lastly, it employs a heuristics based blue agent, although, with modifications, a blue agent can be trained as well.

Another example of a simulator is the **Gym IDS game** that allows for training both the blue and the red agents. The simulator is designed for competitive RL which is modelled as a Markov game and uses self-play for developing security methods that can be used to prevent intrusions [35]. Action space for the attacker agents comprise of reconnaissance and different exploits for specific vulnerabilities. Blue agents can perform the monitor action and various mitigation actions. The observations for each agent are partial, where attacker only observes the machines they control, whereas defender agents do not have any visibility of attacker actions. Similar to CBS, this environment is also based on OpenAI gym interface, that enables it to be used with various RL algorithms.

**Framework for Advanced Reinforcement Learning for Autonomous Network Defense (FARLAND)**, developed by MITRE Corporation, provides both a simulator as well as an emulator, where a set of networked machines are used to train a single blue agent against heuristic based red agents [36]. The simulation layer allows for testing of strategies without the overhead of real network operations, while the emulation is used for validating these strategies. Furthermore, the action space in FARLAND includes a wide range of maneuvers such as reconfiguring network settings, isolating compromised nodes, or even deploying decoy systems to mislead attackers. The observation space provides the defender agent with network activities, from which agents are to find potential threats and mitigate them. Unfortunately, FARLAND is closed source at this time.

**CyBORG** on the other hand is made open-sourced through the Cyber Autonomous Gym for Experimentation (CAGE) challenges [37] and provides both, a simulator and an emulator [15]. Initially, CybORG was released to train a single blue agent against a scripted attacker. However, CybORG is modular and seamlessly allows the red agent to be trained as well. Additionally, with the release of version 3, it enables multiple agents to cooperate for kinetic and network defence scenarios for CAGE challenge 3 purposes, which is obviously a distinct feature when compared to other environments previously discussed. The kinetic scenarios allow users to train agents that are not only responsible for the simulated computer network but also for simulated drones that are connected to the network.

Furthermore, CybORG has a large action space compared to others, that varies depending on the agent's role and the scenario configuration. Some examples of blue agent actions include removing a malicious software from a host, restore if malware cannot be removed, analyse and many more, while red actions are catered for exploits and privilege escalation [15]. The observation space is equally detailed, providing agents with cues into the network's state which also varies based on red or blue's perspective. All the observations are formatted as a dictionary of key-value pairs detailing different aspects of the network and host states. Unfortunately, to facilitate training of RL agents, and to provide the observation space to an algorithm's architecture, the dictionary must be formatted strategically which may abstract a lot of the information. In addition, CybORG is flexible where it can be modified to add more actions, change the basic game scenarios and allows for defender agents to train with real world intricacies such as the green agents which simulate actions similar to normal users generating network traffic. For these reasons CybORG is chosen as an ideal simulator to train agents to communicate in a MARL setting.

## 3.2 RL in Cybersecurity

Historically, the use of RL in cybersecurity has been sporadic, with early applications exploring its potential in a limited scope [8]. With the success of deep learning in RL for various sectors such as gaming [17] and autonomous driving [38], there is an added focus to apply these methods to cybersecurity.

Cybersecurity applications of DRL span various domains, from automating intrusion detection to formulating multi-agent strategies for cyber defence. This section will explore how RL has been integrated into cybersecurity efforts, highlighting its role in developing autonomous systems capable of dynamic decision-making. First it will briefly examine different cybersecurity applications, ranging from IDS to penetration testing and then delineate the strategic deployment of autonomous agents in cyber defence which is relevant to this thesis.

### 3.2.1 IDS

Traditional machine learning techniques have been extensively used for IDS, where normally, a signature based system is developed through supervised learning and an anomaly based system is designed through unsupervised learning [8]. RL frameworks treat the problem of intrusion detection as a dynamic process where an agent (the IDS) interacts with an environment (the network), modeled as a MDP. Within this framework, the state of the network, including various metrics and other observations, defines the environmental states. The agent's actions may involve modifying security settings, conducting deeper inspections, or adjusting network components to trace ongoing attacks [39].

Essential data for training agents in IDS may include network logs for network IDS and specific operating system information for host IDS, which are processed and transformed into a structured format suitable for RL. Network-based IDSs monitor traffic for signs of widespread threats like Distributed Denial of Service (DDoS) attacks, while host-based systems scrutinize activities on individual machines to detect breaches or malware installations [39]. The agents learn through trial and error, receiving rewards or penalties based on the effectiveness of their actions, thereby refining their policies towards optimal intrusion detection.

### 3.2.2 Penetration Testing

Penetration testing (PT) is another area in cybersecurity where RL methods are being investigated. In general PT assesses the security of networks as well as host machines within these networks, by identifying and exploiting vulnerabilities in these systems. Traditional approaches involve manual simulations by testers, which can be inefficient, especially in large and complex networks. Automated tools aim to emulate human testers by applying intelligent strategies rather than exhaustive attack attempts. RL, in this context, adapts well to PT by treating it as a sequential decision-making process where an autonomous agent interacts with a network and learns to perform attacks based on partial observations, adapting its strategy as it gains more information about the network.

A more recent study conducted by Guo et al. explores this space, where they introduce a novel DRL framework to automate the PT process [40]. Their model uses a POMDP framework, where the agents receive partial observations such as operating systems data and existing vulnerabilities. The action space is decomposed into manageable subsets, including actions like local attacks, remote attacks, and connectivity operations. Furthermore, a single agent RL algorithm, Proximal Policy Optimization (PPO) [41], guides the agent's learning process. Moreover, a simulated environment is utilized where the network consists of various nodes with predefined vulnerabilities and each node is exploitable based on a specific vulnerability. Each node possesses certain properties and vulnerabilities, which the DRL agent can exploit.

### 3.2.3 RL Agents in Cyber Defence

Often, the attackers adopt the lateral movement to gain extensive access after breaching a single point within a network [8]. Once they establish a presence on a critical asset, attackers can carry out their goals of compromising the properties of confidentiality, availability and integrity. Also for a simple fact that CybORG allows to simulate these different behaviors of attacker agents, it is intuitive to model cooperation and communication that can enable agents to stop a lateral movement. Thus, this thesis is focused on demonstrating the applicability of communicative MARL in such scenarios and therefore this subsection details a few studies conducted for automating the defence, especially against lateral movements within a computer network.

Prior to recent advancements, the field had encountered a stagnation period due to limitations in available simulation environments. Previously, most research in this area utilized custom-built simulation environments that adopt a simple graph-based framework [42]. In these models, each node represents a host machine within the network, and the connections between them illustrate possible pathways for an attacker's lateral movement [43]. Additionally, the action spaces are simplified as well, where the possible actions are 'to detect threats' and 're-image a host'. While this approach provided a basic structure for studying cyber defence tactics, the simulations often suffered from a lack of realism and practical applicability. It is important to clarify that graph-based frameworks are a foundation for most modern simulators, but the custom-built environments that predate the new simulators were typically oversimplified, failing to capture any complexity and dynamic nature of real-world network interactions.

With the development and open-sourcing of more sophisticated ACO environments in recent years, the research interest in this field has been reinvigorated. The new generation of simulators not only improves the fidelity of the scenarios but also supports the integration of advanced RL algorithms capable of learning and adapting to complex attack patterns in real-time.

For instance, through CAGE challenge 2, CybORG has been utilized to demonstrate the potent of various state of the art single agent RL algorithms in the field of ACD [44]. The challenge is structured in an enterprise network setting which includes multiple subnets with varying roles and importance. The action space for the blue agents comprises of recon, deception and mitigation actions. Moreover, a significant focus is on the hierarchical DRL approach, which uses multiple policy networks with a selector agent to choose the most suitable defensive strategy based on the current threat. The effectiveness of these strategies were measured against various types of red agents, ranging from highly aggressive to more passive ones, which highlights the adaptability of DRL algorithms in ACD.

In a more recent work, Dutta et al. investigate RL algorithms such as DQN and PPO in a cyber defence environment that they developed [45]. The aim of their research is to evaluate the efficacy of various model-free DRL algorithms in adapting to dynamic threats. The environment is based on a multi-stage attack propagation template derived from the MITRE ATT&CK framework. The action space for the defender includes decisions like altering system configurations, implementing security patches, or adjusting monitoring settings to detect and mitigate potential threats. The state of the system is partially observable and the observations include indicators of compromise, network traffic patterns, and alerts that provide information about the potential security breaches or ongoing attacks.

The primary goal of the defender is to avoid the propagation of the attacker through various network stages while maintaining the integrity and availability of the system. The attacker is predefined and is modeled to execute a sequence of actions that evolve as the simulation progresses. The attacker's strategy includes escalating privileges, moving laterally through the network, and achieving their end goals such as data exfiltration or system damage [45].

A recent research is conducted by Nyberg and Johnson, although atypical of the reviewed frameworks of this subsection, but similar in the sense that it trains a blue agent against various scripted attackers to defend a computer network [46]. The simulated environment that the authors used models with attack graphs where nodes symbolize attack steps or defence steps. Attack steps have associated probabilities indicating the time required to compromise them, and the defence steps, when activated, can block the progression of attack steps directly linked to them. The system assumes the presence of an imperfect IDS that provides alert signals. However, the variable detection rate adds a layer of uncertainty. The defender's actions include enabling various predefined defence mechanisms or choosing to take no action. The observation space comprises the states of attack steps, represented by binary indicators of whether each step is compromised.

## 3.3 MARL in ACO

MARL with inter-agent communication is an emerging field and recently there have been a significant amount of research of implementing it for applications in various fields. An area that has attracted some attention is the traffic control at intersections. The problem involves coordinating the movements of multiple vehicles at an intersection to minimize congestion, reduce travel time, and improve safety [47]. Various approaches and algorithms using communication have been proposed, in which an intersection or even a traffic light may be represented as an agent that monitors a number of lanes and a number of vehicles in each lane, share this observation with other agents and collectively agree on the duration of the green light state for each traffic light.

Another area where communication has been applied with some success is resource management. An example is, data centres, where computing resources, such as CPU and memory, are allocated to on-demand applications, albeit inefficiently. In this scenario, multiple agents compete for limited resources, and the goal is to allocate resources efficiently to maximize overall performance [11]. Additionally, cooperative MARL has been realized as a solution in the robot path planning problem. This problem involves finding collision-free paths for multiple robots in an environment with obstacles [11]. Recently there have also been studies in security of Cyber Physical systems such as electric power grids [48], Electric Vehicle Charging Station [49] and other Critical Infrastructures (CI) using MARL.

MARL has also been applied to network IDS, where Shi and He propose a novel approach, called Major-Minor-RL [50] using the Double Deep Q-Networks (DDQN) algorithm [51]. Their model leverages a collaborative multi-agent system consisting of one 'major' agent and several 'minor' agents to improve prediction accuracy in detecting normal versus abnormal network traffic. The environment used for this study is simulated using an offline dataset, which includes a variety of network behaviors classified as either normal or malicious. The major agents have full visibility of the environment whereas the minor agents have smaller areas of responsibility. Each agent is allowed to take an action of classifying if a particular observation is either malicious or benign. The final decision on whether the network traffic is normal or abnormal is determined by a consensus approach where the minor agents can override the major agent's decision if their collective judgment differs.

The most relevant research of incorporating MARL in ACO is the study conducted by Wiebe et al. that applies an independent and a cooperative MARL algorithms, IQL and QMix respectively, to train defender agents to learn tactical level decision making in CybORG [13]. As previously stated, QMix does not use any explicit communication and the coordination is only during training but not during execution. The authors tested the blue agents in confidentiality, integrity and availability scenarios and found that in each case, IQL and QMix are applicable for ACD. Additionally, authors made modifications to the CybORG simulator where they isolated the cyber kinetic aspect and strictly focused on the multi-agent system of an enterprise network. They incorporated this version of CybORG with PyMARL2, a popular MARL library, which they named it CyMARL.

Moving away from cooperative MARL, there are also studies that evaluate agents in a competitive setting. For instance, in a recent work, McDonald examines the application of competitive RL using the CybORG platform [52]. The primary methodology involved developing game designs for red and blue agents, where both agents learn and adapt their strategies in an adversarial setting. The environment is designed to simulate cyber operations, with both agents having distinct roles, observation spaces, and action spaces. Furthermore the author chose a setup where agents use policies based on the Fictitious Play algorithm [53]. This approach allows both agents to continuously adapt by considering the average policy of their opponent, which evolves as each agent updates its strategy through iterative learning cycles. The aim is to reach a Nash Equilibrium, where neither agent can unilaterally improve their policy without decreasing their performance against the other's strategy.

## 3.4 Summary

This chapter detailed various ACO environments, with a key finding that simulation environments abstract away a lot of intricacies of real networks that may potentially hinder deployment of autonomous agents in a real world setting. This chapter also reviewed various studies of cybersecurity problems, including the field of ACO, which highlights the potential of RL. Lastly, MARL research is discussed, where it is found that very limited studies have been conducted with applying multi agent systems in cybersecurity. To the best of our knowledge, there has yet to be an empirical study of communication algorithms in cooperative MARL for the problem of ACD.

# 4 Methodology

This research unfolds across three distinct phases as seen in Figure 4.1, where the composition of the experiments in each phase is influenced by the evaluation of results from the previous phase. This chapter outlines the methodology employed at each stage of the research and how these experimental procedures evolve in response to the iterative analysis of results. The next chapter will then detail the evaluation and results derived from these trials.

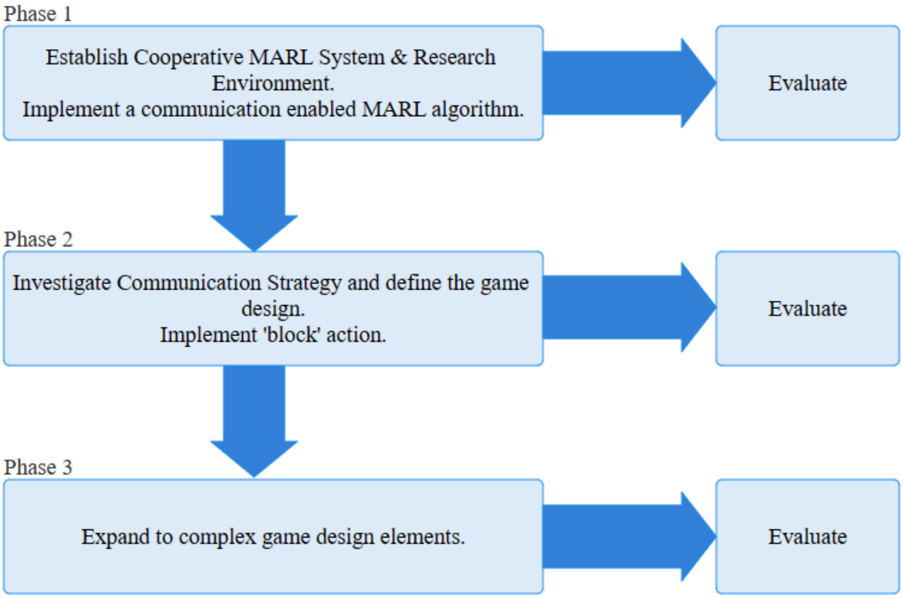

Phase 1
Establish Cooperative MARL System & Research Environment.
Implement a communication enabled MARL algorithm.
Evaluate
Phase 2
Investigate Communication Strategy and define the game design.
Implement 'block' action.
Evaluate
Phase 3
Expand to complex game design elements.
Evaluate

**Figure 4.1:** Research Activities.

In Phase 1, the CyMARL environment is adopted, where a confidentiality scenario is selected from various available scenarios and a baseline is established. A communication enabled MARL algorithm is then integrated within the established environment. Subsequently, agents are trained and evaluated after the integration of the algorithm. Based on the analysis of results from this evaluation, the base scenario is reconfigured to enable agents to convey meaningful messages to each other which entails Phase 2 work. Furthermore, more complex game design elements are explored where a new action is introduced in the scenario allowing the agents to stop the adversary from lateral movement. Lastly, Phase 3 of this research introduces new and more complex scenarios with the established game design to test the scalability of the learned communication strategies..

## 4.1 Establish the Research Environment

As stated earlier, CyMARL incorporates CybORG, the cyber defence simulator environment for RL [15], and PyMARL, a MARL library [14]. The DIAL algorithm is not incorporated within PyMARL. As such, in order to train the defender agents to communicate, the algorithm is integrated separately. A design decision needed to be made because although it is possible to utilize the default version of CybORG and modify it with the communication algorithm, it was a better to proceed with CyMARL for two reasons: 1) using QMix in PyMARL to train agents for baseline, and 2) for the fact that CyMARL code-base consists of a utility that allows for multi-agent scenarios in CybORG.

Furthermore, the use of a command-line interface provides a simplified process for training agents. Users are able to start the training sessions with specific configurations of the environment and choose the reinforcement learning algorithm through command-line arguments. This approach eliminates the need for extensive coding, making it accessible to users with varying levels of expertise and programming knowledge.

Additionally, to conduct research that explores wide range of cyber defence strategies, CyMARL enables users to experiment with algorithm parameters, adjust the network architecture of agents, modify environment variables, and simulate different attack scenarios using simple human readable configuration (YAML) files. Table 4.1 lists some important parameters that were applied for the baseline.

| **QMix configuration** | | **Environment Configuration** | |
|---|---|---|---|
| **Parameter** | **Value** | **Parameter** | **Value** |
| Batch size | 128 | Total timesteps | 19.2M |
| Buffer size | 10000 | Episode limit | 30 timesteps |
| Learning rate ($\alpha$) | 0.001 | Scenario | Confidentiality |
| Rollout size | 8 | Wrapper type | Vector |
| Discount Factor ($\gamma$) | 0.90 | Action Masking | False |
| RNN hidden layer dim | 64 | Reward Calculator | Confidentiality |
| Target update interval | 200 | | |

**Table 4.1:** Parameters used to train agents in baseline [13].

Moreover, the results of each experiment, including the performance metrics and learned policies are automatically saved in a specific directory. The framework supports evaluation on trained models that can be used to test against existing or new attack scenarios. This feature is especially important to assess the robustness and generalizability of the learned defensive strategies. In addition, a utility is provided that aids in visual analysis of training progress and outcomes in real time. Different types of plots and charts can be generated utilizing the visual analysis tools that shows agent performance over time.

### 4.1.1 Cyber Operations Research Gym

CybORG, at its core, is heavily influenced by game theory principles with the defensive challenges it presents for cyber operations. The environment is designed to simulate complex interactions between blue and red cyber agents, where they engage in a continuous strategic game. While it is possible to train both, offensive and defensive agents, this research is focused on the cooperative strategies of defender agents and therefore, the red agent is predefined.

CybORG is distinguished by its modularity and diversity of scenarios it supports, ranging from simple network configurations to complex enterprise-like simulations. These scenarios are outlined using YAML files and define the rules of the game such as the role of participating agents, the zones they control, their available actions, their initial knowledge, and the criteria for success. The file also dictates the configuration of host and network connections, simplifying the environment setup by employing minimal information requirement. A scenario generator utility then takes these configurations and initializes the environment where each element reflects the real-world intricacies.

At the onset of each training session, an environment controller is created, that keeps track of the true state of the game. This state contains network configuration information as well as all the details pertaining to hosts that are reset at the beginning of every episode of the game. Data such as name of the host, host ip address, host-subnet mapping, and active sessions of each agent on all hosts are part of the true state. These are maintained and updated as blue and red engage in this gym cyber battle. Only partial information from this true state are passed down to each agent, red and blue, which helps them in determining their next action.

Moreover, CybORG simulator is designed to be a host-based system, where the environment enables the agents to learn host-level defensive tactics. These tactics are tailored to the unique characteristics and vulnerabilities of each host within the simulation. The realism of the simulated environment is further enhanced with the variety of host images that CybORG supports. Various Linux distributions and Windows versions are incorporated, where each host image comes with its own unique set of vulnerabilities, mimicking the heterogeneity found in actual cyber ecosystems. In addition, these images are detailed to include a variety of processes and the specific ports they operate on as well as the user permission settings, complete with usernames and passwords for both standard and administrative accounts.

Finally, wrappers are provided in the environment, which play a pivotal role in interfacing between the agents and the simulation environment. A wrapper is essentially a tool that wraps the environment with features that ensure compatibility of RL algorithms without having to modify the underlying code. One such wrapper is responsible for constructing actions for the blue agents, translating high-level defence strategies into specific, executable commands within the CybORG environment. Another critical wrapper translates the complex dictionary of observations returned by the scenario into a vector form. This conversion is essential for making the data machine-readable, thereby enabling the algorithms to process and learn from the environment's state effectively.

### 4.1.2 Network Configuration

The network architecture for the cyber defence simulations in this research is depicted in Figure 4.2. The setup involves six hosts across two subnets, where two blue agents, each overseeing a separate subnet, work to mitigate the advances of a red attacker agent. At the start of every episode IPv4Network and the IPv4Address for each subnet and host respectively, are randomly generated along with the routing information to other hosts. Each host in the network has a

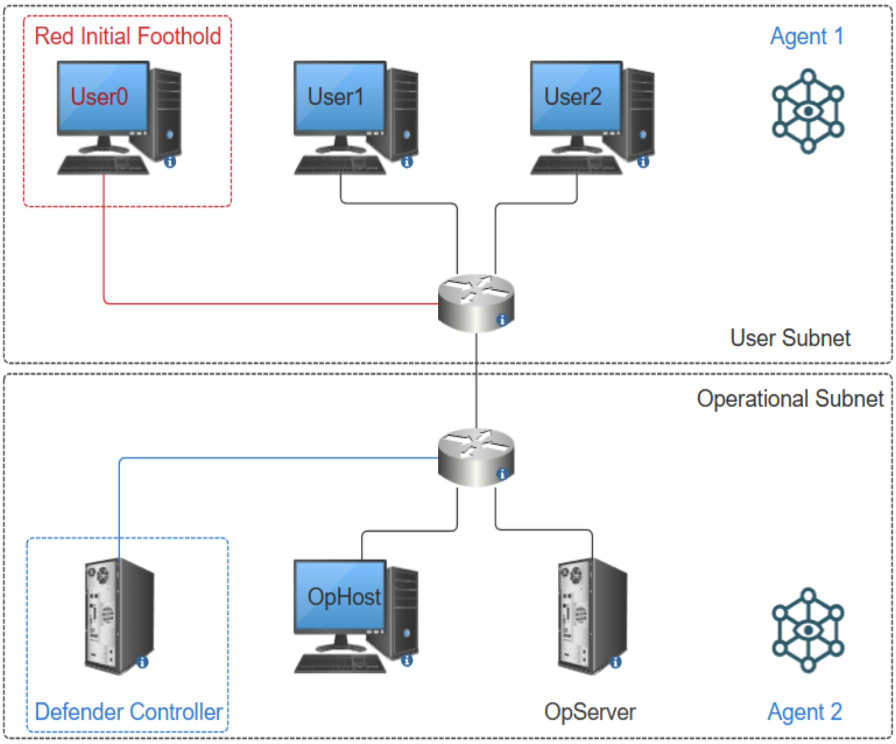


**Figure 4.2:** Network established for cyber games in CybORG.

specific service running, which the attacker aims to exploit. Each host also has a defender session that is administered by a controller (Defender Server) and allows the blue agents to perform mitigating actions. This foundation forms the basis of the scenario that is leveraged to train the blue defender agents. The network comprises:

- **User1**: Operating on Windows Operating System (OS) and exposing a Secure Shell (SSH) service on port 22, this host is susceptible to SSH brute-force attacks, representing a common vulnerability point within the network.

- **User2**: Running on Linux OS, this host has multiple processes running across various ports, each associated with exploitable vulnerabilities. Critically, User2 holds a unique interface to the operational subnet, enabling the red agent, upon achieving admin escalation, to uncover the subnet's network address. After this discovery, the red agent can initiate reconnaissance for IP addresses and open ports within the operational subnet.

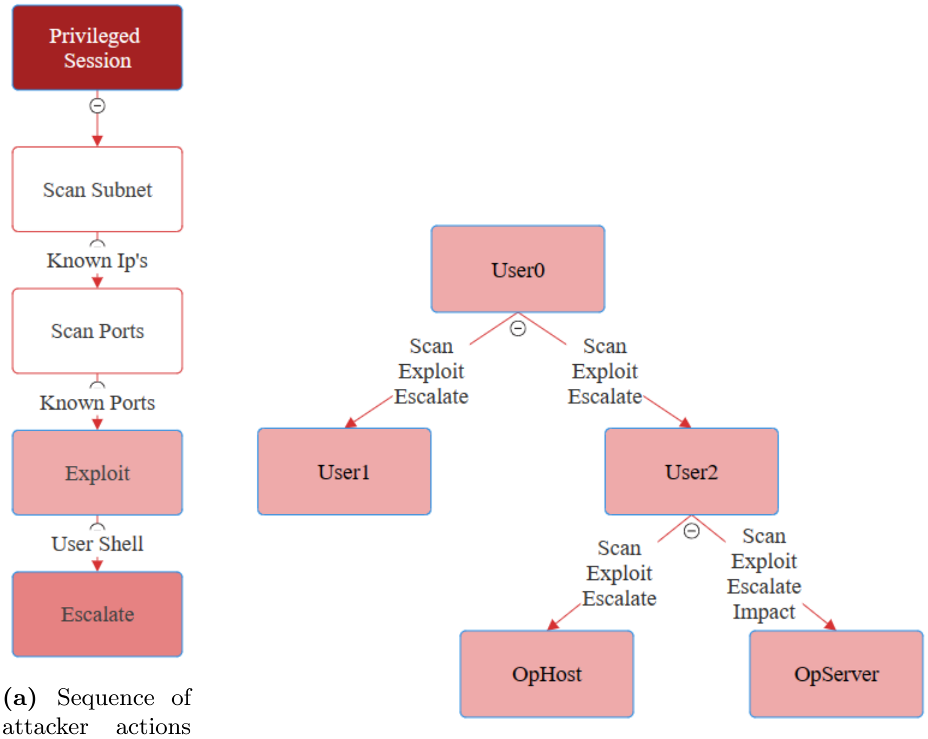
- **OpHost**: A Linux-based system offering a SSH service. With the open port 22, it is also vulnerable to brute-force attacks.

- **OpServer**: Similar to OpHost, it is also a Linux-based system with an open SSH port. This server is a critical asset and also the main objective of the attacker. A service is running on this server, where once escalated, a simulation of stopping this key service can be executed by the red agent.

- **User0**: This host is shielded from blue agents' field of view and is integral to simulating the red agent's initial network foothold. Although this windows-based host has many vulnerabilities and open ports, it is modelled such that blue agents cannot remove the red agent's session from this host.
- **Defender**: This server acts as the operational base to both the blue agents and by design remains invulnerable to red agent's attacks throughout the game. This ensures the uninterrupted functionality and response capability of the defence mechanisms within the simulated network.

### 4.1.3 Attacker Agent

The attacker agent begins each episode with a strategic foothold within the user subnet, specifically on User0. This initial presence sets the stage for a series of attacks, where the agent systematically targets each host based on the intelligence gathered throughout the scenario. The red agent follows a predetermined attack path designed to exploit vulnerabilities and escalate privileges, methodically working its way towards the operational subnet's critical server. The sequence of strategic decisions made by the red agent is illustrated in Figure 4.3. As can



**(a)** Sequence of attacker actions on a host.

**(b)** Trained policy for agent 2 with block action.

**Figure 4.3:** Attacker agent's decision tree in the base scenario.

be observed, once User2 is escalated, red gains a crucial advantage, effectively bridging the gap to the operational subnet.

The red agent has many actions in its repertoire, detailed in Table 4.2, and are available throughout an episode. Initiating its offensive from the initial session, the agent commences with an IP address sweep in the user subnet. Upon acquiring the IP addresses, it proceeds to scan them for any open ports. Identifying a vulnerable port, the agent then randomly attempts to compromise a host. The success for spawning a shell is contingent on a combination of factors such as the specific exploit utilized as well as the service that is associated with the targeted port. After obtaining a user shell, a deliberate random delay of 2-3 timesteps is imposed before advancing to escalate the host's privileges. This intentional pause not only allows the agent to extend its control to other hosts, but also injects a degree of unpredictability in its attack pattern, which complicates the defensive strategies of the blue agents.

| **Action** | **Description** |
|---|---|
| DiscoverRemoteSystems | Action that performs a pingsweep on a particular subnet. Requirement is to know the IPv4Network. |
| DiscoverNetworkServices | Performs a portscan on a particular IP address found after performing the above action. |
| ExploitRemoteService | This requires an IP address and a port identified from the above 2 actions after which the attacker will be able to exploit a known vulnerability. |
| PrivilegeEscalate | Allows the attacker to escalate to admin after a vulnerability has been exploited. |
| Impact | This action can only be performed on the operational server, where once escalated, it simulates stopping of a key service on the server. |

**Table 4.2:** Actions available to the attacker agent

Additionally, should the defender agents succeed with their mitigation actions, the red agent will attempt to reestablish its sessions on these hosts. Similarly, after the discovery of the operational subnet, red follows the same sequence of actions and attempts to gain admin privileges on the operational server. Lastly, red agent can 'impact' to stop a key service provided by the server. The decision making process that governs the red agent's actions is shown in Algorithm 3.

The default scripted attacker leverages the session established on User0 for targeting hosts across both subnets. Consequently, once red discovers the operational subnet, it no longer requires the session on User2 for recon and exploit actions on hosts in the other subnet. All pertinent data, including IP addresses and their corresponding open ports are saved in the initial session. As a result, the attacker navigates its way to the operational server, undeterred by any countermeasures deployed by the blue agents. This is especially discouraging in blue agent's ability to learn to thwart the red's lateral movement. For instance, even if blue manages to remove the red's session on User2 following its compromise,

**Algorithm 3** Scripted Attacker Agent

```
Initialize: discovered_subnets, scanned_subnets, discovered_ips, scanned_ips,
  exploited_ips, escalated_hosts
function GetAction(observation, action_space)
    Process success of the last action and update internal state
    if Op_Server in escalated_hosts then
        return Impact action on Op_Server
    end if
    for each subnet in discovered_subnets do
        if subnet not in scanned_subnets then
            return DiscoverRemoteSystems action for subnet
        end if
    end for
    for each IP address in discovered_ips do
        if IP not in exploited_ips or scanned_ips then
            return DiscoverNetworkServices action for IP address
        end if
    end for
    for each IP address in exploited_ips do
        if privilege escalation delay met and not in escalated_hosts then
            return PrivilegeEscalate action for host
        end if
    end for
    for each IP address in scanned_ips with open ports do
        if IP not in exploited_ips then
            return ExploitRemoteService action for IP address
        end if
    end for
    return Sleep action if no other actions are applicable
end function
function ProcessSuccess(observation)
    Update internal tracking based on the success of the last action
end function
```

red has an easy path without any defence to the critical asset due to the cached IP addresses and ports. From a real-world standpoint, this is an acceptable behaviour if there are no access restrictions between subnets. However, a more logical and realistic approach would be to use the session established on User2 as a springboard that moves the attacker one step closer to its goal.

A minor but pivotal modification to the default scripted attacker enables the red agent to use the session that is closest to its target. In this modified scenario, following the escalation of User2 and the discovery of the operational subnet, the actions to move into the other subnet are executed from the User2 session. This adjustment ensures that if the blue agents intercept the attack before red advances to the operational subnet, the attacker must regroup from the initial foothold, re-compromising necessary hosts to progress. This logic significantly influences the defender agent's learning curve and strategic responses. The scenario makes an important assumption in implementing this strategy, that User2 has some specific connections to the interface of the operational subnet.

### 4.1.4 Defender Agents

The goal of the scripted attacker is to infiltrate the operational server and disrupt the essential services it provides to its network users. In response to this behaviour, blue agent's objectives can be articulated as preventing the attacker's advances to the server or mitigating the impact if the server is compromised. Ideally, the mandate for each defender agent is simply to learn defensive tactics that effectively confines the attacker to its starting point. This demands not only timely and appropriate defensive responses but also a concerted effort between agents. The reward function plays a pivotal role in guiding the blue agents towards optimal behavior. It encourages actions that effectively neutralize threats and penalizes those that fail to secure the network. Lastly, the observation space equips the agents with insights into the network state, that enables the agents to make informed decisions.

**Environment Actions**: Defender agents can perform a variety of actions to achieve the goal at protecting the network. These actions are listed in Table 4.3 and fundamentally are two types of actions: passive and active. In this scenario, 'monitor' is devised to be a passive action that is executed at the end of each turn. This action can be conceptualized as a host-based IDS that generates alerts in response to multitude of connections between hosts or when processes are created or tampered with on a host. These indicators signal the agents of a potential malicious activity that may require intervention. While the 'monitor' action is passive, it could be regarded as an active measure that require the agents to learn to take it appropriately. Especially within the context of a larger enterprise network and with a significant amount of network activity, it is infea-

| **Action** | **Description** |
|---|---|
| Monitor[1] | Allows the defender agents to detect port scans and exploits on hosts within its field of view. |
| Sleep | Do nothing. |
| Remove | Removes any suspicious processes from any host within its visibility. |
| Restore | This action allows the defender agent to re-image and restore a particular host. |
| Analyse | Defender agents can use this action to identify malicious files on a host. |
| Block[2] | Allows agents to block any traffic from a particular subnet. |

[1] Passive action available at the end of every turn. [2] Action available in Phase 2.

**Table 4.3:** Actions available to the defender agents.

sible to monitor every single host passively. Nonetheless, given the focus of this research to enable agents to learn a communication protocol within a relatively contained network, it is appropriate to provide the agents with a comprehensive state information at every step.

The remaining actions are implemented as active measures and therefore are required to be learned. Upon discovering a threat on a host, a suspicious process ID is identified and added to the blue agent's session. The defender then can use the 'remove' action to kill this process, which removes the red's session on that host. This means that the 'remove' action will only work after a successful discovery of an exploit. It will also not function after red has escalated its privileges. The assumption here is that once red has gained a privileged shell, it is persistent and therefore a process cannot be killed at this stage.

'Restore' action is essential for blue agents if a 'remove' action is not successful. By using the 'restore' action, a system can be reverted back to its initial state which allows the removal of all the attacker sessions on the host. This action causes some disruption on the network in particular to the server that is providing key services and therefore is designed to function only on hosts with known or suspected threats. Moreover, 'block' action is the only action that is not applicable to a host but to a particular zone (subnet). This action allows an agent to block any traffic for one timestep that is initiated from a singular subnet. This is primarily useful to the agent responsible for the operational subnet in averting an imminent attack on its hosts.

Furthermore, the passive 'monitor' action is able to detect a user-level shell on a host. However, it is not able to detect an elevated shell because once red escalates a system, it is obfuscated from blue's observations. The only way blue can determine this is by using the 'analyse' action. A successful escalation by red results in a malware file being left on a system and with the use of the 'analyse' action, blue agents are able to find these malicious files. This action is an abstraction of malware analysis tools such as DensityScout, which measures the entropy of files, and SigCheck, which verifies the signature of a file.

All defender actions are parameterized which means that the blue agents require specific information to execute the actions. Firstly, blue agent's session ID is a common parameter for all actions. Each agent that is responsible for a subnet has an active session running on the defender controller and the agent that is executing an action must use its own session. An IP address is also required to perform actions on a specific host. The 'block' action is the exception, as it does not require an individual IP address but instead needs an IPv4 network address to block an entire zone. Lastly, the 'remove' action requires an extra parameter, which is the process ID of the suspicious process identified by the 'monitor' action.

**Reward Function**: The reward function is intricately designed such that there are no positive rewards but only penalties associated with any failures in defence. It does so by assigning negative rewards that reflect the importance of different hosts within the network. It is also devised to impose costs for unsuitable action or actions that cause disruption to the network, thus guiding agents towards a prioritized defence strategy.

The reward function does not assign a penalty or a reward for successfully removing a malicious process or for finding a privileged shell through the use of 'analyse' action. Although experiments were conducted where positive rewards are assigned for these two specific scenarios, there is no added benefit with the positive rewards in the agents ability to learn. This is due to the fact that any privileged red session that stays active will accumulate the penalties at each timestep and both agents must mitigate the exploits to minimize these penalties. The function is outlined as follows:

- Penalties for Host Capture (Escalation by Red): If the attacker successfully escalates privileges on a host, the defender incurs a penalty proportional to the host's importance within the network.
  - User Subnet Hosts: Being of lower importance, each captured host in this subnet results in a minor penalty of -0.1, signaling a breach but recognizing its relatively lower strategic value.
  - Operational Host: This host carries a medium importance value, and the capture of this host results in a more significant penalty of -1.0, indicating the attacker's progression towards the server.
  - Operational Server: As the most critical asset, the operational server's compromise leads to a severe penalty of -10.0, reflecting its high importance and the consequent impact of its loss on the network's overall security posture.
- Penalties for Restore Action: When a 'restore' action is performed, the penalty is once again based on the importance of the host being restored which is -0.1, -1.0, and -10.0 for low, medium, and high importance hosts, respectively. This encourages defenders to prioritize the recovery of more critical systems and emphasizes the importance of swift recovery actions in maintaining network integrity.
- Penalties for Inappropriate Actions: The defenders are also penalized for actions that do not contribute to the network's security, for example, unnecessary or ineffective use of 'remove' and 'analyse', with a standard penalty of -0.5. This ensures that the agents learn to perform only strategic actions.
- Block Penalty: To discourage excessive blocking actions, which could disrupt network functionality, a flat penalty of -1.0 is imposed for each 'block' action.

This reward function serves as a feedback mechanism that teaches the defenders to distinguish between the varying degrees of priority among the network's assets. Likewise, it equips the agents to minimize the damage caused by red's presence on low priority hosts, where penalties are accrued over time. Additionally, given the fact that the red agent penetrates the high-valued server through an unimportant host and since the penalties are significant for the operational systems, the reward function also enables the agents to adapt their priorities.

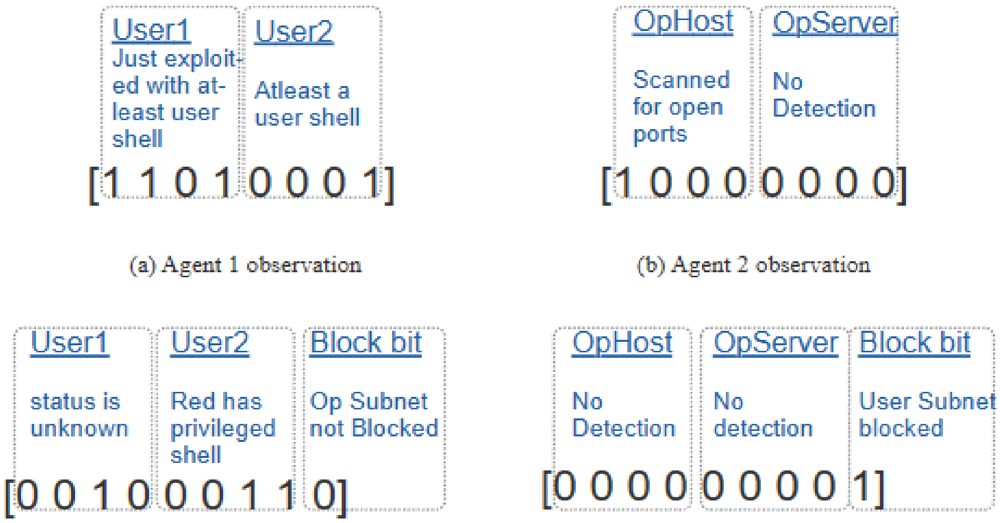
**Observations**: As previously stated, CybORG returns raw observations in the form of a python dictionary which provides information that is related to the previous actions undertaken by both the red and blue agents. The dictionary encompasses details pertaining to each host such as network interfaces, processes and system details. Suitable information from the output of these raw observations is transformed into a binary format that are meant to aid the blue agents to learn cyber defence tactics. As an example, a combined observation space for each agent represented as a vector is shown in Figure 4.4, where each host's state is described by four binary digits. The first pair of bits portray recent activity of the red agent while the last two bits denote the current status of the host, detailed as follows:

Activity indicators:

- [0,0]: No activity detected
- [1,0]: System was scanned for open ports in the previous timestep.
- [1,1]: Host was exploited in the previous timestep.

Status indicators:

- [0,0]: No threats detected on the device.
- [1,0]: Status is unknown.
- [0,1]: Red has at-least a User shell.
- [1,1]: Red has a privileged shell on the host.



(a) Agent 1 observation

(b) Agent 2 observation

(c) Agent 1 observation with block bit

(d) Agent 2 observation with block bit

**Figure 4.4:** Observation spaces for blue agents in the base scenario.

Additionally, the raw observations contain a success factor for each blue agent's active action, indicating whether an action achieved its intended outcome. This aspect refines the observation vectors, enabling the blue agents to discern the impact of their decisions within a given state. Furthermore, in Phase 2 trials, an extra bit is introduced to signify whether a 'block' action is executed in the preceding timestep.

Moreover, it is impractical to monitor every aspect of a network comprehensively, especially given the covert nature of many cyber attacks. Acknowledging this, CybORG is designed to realistically simulate the detection conditions by ensuring that all exploits, that are not ssh brute force attacks, are detectable with a 95% rate. Furthermore, the observation vector is not a complete representation of the true state. For example, agents are not aware of any reverse shells that are privileged, nor do they have information on host status or red activity in other subnets. Nonetheless, the partial observations play an important role in enabling the blue agents with the necessary situational awareness to effectively defend the network. The observations serve as the primary input to the algorithmic architecture, which is designed to guide each agent's decision making process.

## 4.2 Algorithm Implementation

Several state of the art algorithms, discussed in 2.3, were considered for this research within the CybORG environment. The DIAL algorithm distinguished itself as the best selection due to its simplicity, efficiency, and emphasis on interpretable and discrete communication mechanisms. Although, all algorithms that were examined allowed explicit messages to be sent, DIAL facilitated an environment where the communication is external to the neural network architecture. For instance, models such as CommNet, BiCNet and IC3Net has communication embedded within the network, obscuring the content of inter-agent messages. This means that only partial observations are provided as inputs to the network, while the communication occurs within the hidden layers without any strategic message outputs. On the contrary, in DIAL, messages from each agent are provided as specific inputs to the network, leading to distinct communication actions alongside environment actions.

Additionally, CommNet and BiCNet utilize a central controller that receives individual agent messages and disseminates them among agents. This centralized approach is a potential bottleneck and risks efficiency losses, particularly in scenarios requiring rapid and scalable responses to dynamic threats. DIAL's methodology sidesteps this issue by enabling direct, agent-to-agent communication. Furthermore, CommNet, TarMAC and IC3Net allow a multi-step communication process within each timestep which introduces an additional layer of complexity to training. In contrast, DIAL's single-step communication model aligns better with the decision-making in cyber defence.

Lastly, CybORG simulations are designed for homogeneous agents. Homogeneous agents in a multi agent system refers to all the agents in an environment that fundamentally have identical functionalities and characteristics, while heterogeneous agents have diverse set of actions and features. IC3Net is designed to handle heterogeneous agent scenarios and although the approach can be applied in CybORG, it does not offer any substantial benefits.

### 4.2.1 DIAL-CybORG Integration

The authors of the DIAL algorithm provided a version of their implementation[1] that can be used to train agents for the switch riddle game. This not only offered a foundational model to imitate switch game's communication mechanisms but also provided a code which, with a few modifications, can be adapted for agent communications in CybORG. Firstly, agents in CybORG were conditioned to communicate based on specific observations on their monitored hosts. This eliminated unwanted communication that can be regarded as noise to other agents. This also mirrored the specifics of the switch game where communication occurred in a constrained environment (e.g., within a room). Additionally, communication is limited to a 1-bit message, similar to the switch game's communication strategy. Despite DIAL's capacity for more complex messages, the 1-bit approach, validated through experimentation, proved more conducive to learning within the abstracted context of CybORG.

Moreover, to bridge the gap between CybORG's outputs and the DIAL algorithm's inputs, a critical utility is designed. This utility ensured that environment outputs such as observations, rewards and terminal information were formatted correctly into data structures suitable for the algorithm. This utility, along with the use of the 'action wrapper', also transformed the high level actions output by the algorithm into specific commands in the environment. Furthermore, utilizing the methods in CyMARL, parallel processing is enabled that significantly provided performance boost of the integrated system.

As discussed in Section 2.4.1, a task embedding $z_t^a$ is produced from four inputs from the current timestep which is passed to the RNN. To produce $z_t^a$ at each timestep, the four inputs are summed element-wise as seen below:

$$\begin{aligned} z_t^a = (TaskMLP(o_t^a) + MLP[|M|, 128](m_{t-1}) + Lookup(u_{t-a}^a) \\ + Lookup(a)) \end{aligned} \tag{4.1}$$

The $TaskMLP(o_t^a)$ is the embedding for the partial observations from the environment and is dependent on the type of observations the environment produces. For instance, in the case of the switch riddle experiments performed by the authors of DIAL, the $o_t^a$ are passed through a lookup table which produces this embedding of size 128. The message input $m_{t-1}^{a'}$ is passed through a 1-layer MLP, and inputs $u_{t-1}^a$ and $a$ are passed through lookup tables all producing an embedding of size 128 [25].

The message, previous action and agent identifier inputs were unchanged from the switch game implementation. However, the complex observation space of cyber defence scenarios in comparison to the switch game proved to be challenging. Due to the finite state space associated with each host, a systematic technique is employed to represent a host. These were further summed element-wise to obtain the final embedding of the partial observations of each agent which is illustrated

[1] `https://github.com/minqi/learning-to-communicate-pytorch`

below:

$$\begin{aligned} z_t^a = ((Lookup(host\_1_t^a) + Lookup(host\_2_t^a)) + MLP[|M|, 128](m_{t-1}) \\ + Lookup(u_{t-a}^a) + Lookup(a)) \end{aligned} \tag{4.2}$$

### 4.2.2 DIAL Hyperparameters

Hyperparameters are configuration settings that are used to structure the learning process of machine learning models. These values are set prior to training. They cannot be changed during the learning process and have a significant influence on the model's ability to learn patterns from the data [54]. Proper adjustments of these parameters allows for increased performance and accuracy of the model. Although there are many techniques available to fine-tune these parameters, it is considered to be out of scope for this research. And therefore, the exploration of hyperparameters is targeted rather than exhaustive, with a primary focus on refining game design over optimizing algorithmic efficiency. Table 4.4 below outlines the key hyperparameters and their configured values.

**DIAL configuration**

| **Parameter** | **Value** |
|---|---|
| Batch size | 128 |
| Rollout size | 8 |
| Learning rate ($\alpha$) | 0.0005 |
| Discount Factor ($\gamma$) | 0.90 |
| Exploration rate start, finish ($\epsilon$) | 1.0, 0.05 |
| Exploration anneal time | 1M timesteps |
| RNN hidden layer dim | 128 |
| Target update interval | 100 Epochs |

**Table 4.4:** Parameters for the DIAL algorithm.

Nonetheless, a few parameters were experimented with, which are outlined below:

- Batch Size: This determines the number of samples processed before the model is updated. A relatively larger batch size of 128 is chosen based on experiments that indicated better performance compared to smaller batches (32, 64). It may have been possible to further improve the performance with a higher value, but this would significantly extend the training duration.

- Learning Rate: Learning rates of 0.001, 0.0005 and 0.00025 were experimented with and the 0.0005 proved better in stabilizing the learning process. The parameter impacts the convergence speed and stability, where a higher rate accelerates learning with a risk of overshooting the global minima and a slower rate significantly lowers the magnitude of the updates.

- Exploration Rate: In the switch riddle, the authors applied a static exploration rate of 0.05, which proved sufficient as the agents in that game could only take 2 possible environment actions at any given time step. There are 7-8 possible actions per agent in the CybORG environment and therefore, adjustments were necessary to accommodate the increased action complexity. Each training session started with an exploration rate of 1.0 (pure exploration) which is annealed to 0.05 over 1 million timesteps which facilitated a more balanced exploration-exploitation trade off.

- Discount Factor: A discount rate of 0.9 is selected for the experiments. The discount factor signifies the importance of future rewards and is especially important in CybORG where a miscalculation in defending a low priority host can lead to a compromise of higher-valued asset with harsher penalties.

## 4.3 Communication Strategy and Game Design

Originally, CybORG was designed for single-agent scenarios where an agent is enabled to oversee an entire network. This implies that the abstracted environment might not require more than one agent for its autonomous defence. Indeed, it has been demonstrated through CAGE challenges that sophisticated single-agent algorithms are capable of securing the network across diverse scenarios [44]. Furthermore, as highlighted by Wiebe's research, the CybORG game can be successfully navigated by multiple agents without the need for coordination. This suggests that, in its current form, CybORG does not inherently necessitate communication between agents.

The capacity for resolving the game through independent agent strategies is further affirmed in the initial trial phase with the DIAL algorithm. Notably, in the base scenario where 95% detection rate is used on hosts susceptible to non-SSH brute-force attacks, agents independently formulate effective strategies to counter threats before they jeopardize critical assets. Nonetheless, the goal of this work extends beyond simply integrating a communication algorithm. The primary focus is to leverage the DIAL algorithm's capabilities for fostering meaningful agent communication and ensure that the network's defence is coordinated.

### 4.3.1 Port Scan

In a real-world setting, port scanning is a critical activity in network security, that not only serves as a diagnostic tool for administrators but also a precursor to potential attacks [55]. Generally, benign users are not allowed to scan hosts for open ports, especially those in operational networks and vital systems. And even if scanning is required, it is not done without proper permissions and prior coordination. Therefore, port scanning is typically viewed as a hostile action and signals an increased likelihood of an impending security breach.

During trials in Phase 1, it was noted that the defender agent in operational subnet actively responded to instances of port scanning. This agent would send alerts to the other agent in the user subnet, indicating possible compromise of a host. This strategy proved effective in the early to middle stages of training, where the receiving agent reacted promptly to mitigate any threats. However, as the trials progressed, the agents gradually shifted away from using this communication strategy. Communication became less meaningful as agents learned to independently and efficiently devise their defensive strategies. Regardless, these initial communication policies paved the way to solidify the game design elements for future experiments. The details of the analysis is reserved for Chapter 5.

### 4.3.2 Detection Rate

As noted, a relatively high detection rate of 95% is used in Phase 1 trials. This means that in the latter stages of these trials, where agents are close to converging on optimal defensive strategies, they rarely encountered scenarios where the intruder initiated its recon and infiltrated the operational subnet. The agent behavior in reinforcement learning is fundamentally shaped based on the data the learning model receives. The samples with high detection provided to the model naturally skewed the policies towards non-communicative and independent. Additionally, it is uncharacteristic of real-world scenarios where a high certainty in threat detection is rare.

To address these limitations and to introduce more realism, a scenario with reduced detection rate of 50% is setup. This adjustment would enable the agent in the user subnet to autonomously mitigate threats on User2 host in half of the instances. For the other 50%, where the red agent manages to covertly elevate its privileges on User2 host and initiate recon on the operational subnet, the corresponding blue agent must promptly alert its counterpart. Initially, this strategy proved to be effective, where agents leveraged the communication channel to manage threats. However, over time, agents again veered away from meaningful communication.

This evolution of the defensive tactics can be attributed to the recurrent neural network architecture of the DIAL algorithm. RNN, which has a memory mechanism, allows the agents to minimize the penalties by adequately timing their responses even without detection. Basically, the agents, after observing the game for a few thousand episodes, are able to anticipate the compromise of hosts without any valuable information received from their counterparts. To some extent, this is also true in the case of the QMix algorithm, which is used as benchmark against DIAL and also uses RNN for its agent network. Similar to DIAL, agents in QMix are not aware of host status in other subnets within its learned decentralized policies, but are able to effectively mitigate threats in same conditions. While this technique is efficient within the CybORG framework, its applicability in real-world scenarios, with more variable and less frequent attacks, is limited. To account for this drawback, a technique known as action masking which is discussed below, is devised that limits the environment actions to be used only after specific observations.

### 4.3.3 Action Masking

Masking of invalid actions in RL has been widely utilized in many simulators that have a large discrete action space. For instance, in the StarCraft II challenge, which has over a hundred actions, it is used to limit the invalid actions and refine the learning process [56]. The technique is implemented by restricting bad actions in a given condition, within the environment, and not allowing an agent to execute it. This enables the model to only process and learn certain actions in any given state. Studies have shown that this method significantly accelerates the learning process, which facilitates agent's attention on viable actions and reduce the exploration space [57].

In value-based learning frameworks, for exploitation, the agent selects its actions based on the highest predicted Q-value. With action-masking, the Q-values are set to 0 for all irrelevant actions, ensuring that the agent only considers permissible actions. Meanwhile, during exploration, an action is randomly selected from the list of only available actions.

For the CybORG environment, action masking is implemented for the 'remove', 'restore', and the 'analyse' actions. These actions are now contingent upon the detection of a threat on a host and serves many purposes in streamlining the learning process. Firstly, 'remove' action would fail anyways and be rendered illegal without prior detection. This is due to the fact that a malicious process is not identified on a host where a reverse shell is undetected. Secondly, the 'restore' action, could technically be executed on hosts with undetected compromises. However, restricting it to be used on hosts with known threats prevents unnecessary disruptions especially on the operational systems. Lastly, restricting the 'analyse' action reflects upon some real-world considerations such as time and resource cost associated in using this action. The availability of this action is conditioned based on two criterias: 1) Similar to 'remove' and 'restore', it is available when detection occurs, and 2) without detection, it can be only used after inter-agent communication. The core idea and assumption here is that the operational subnet is not directly accessible to malicious actors. This is normally the case with enterprise networks where the architecture is segmented and has multiple layers of security that place the essential systems much deeper within the network. Therefore, once front facing segments are penetrated, attackers have to probe deeper to get to the controlled environment. With this, the defender agents can deduce that the intrusion into the critical zone must be from the user networks. Therefore, even if red has covert sessions in a user network, the operational agent is able to send a signal to its counterparts, alerting them of some malicious activity. Obviously, for this to function as described, there has to be some indicators, with 100% confidence, within the critical zone. Although port scanning in a real-world situation may not be sufficient and not identified at all times, it is strategically used in these scenarios to provide the certainty that is required.

Within the DIAL algorithm architecture, messages are directly sent from one agent to another where the algorithm is designed to not allow the communication to flow into an environment. However, to facilitate the strategic use of the 'analyse' action, the message output from an agent is allowed to filter into CybORG. A received message, during both learning and execution, is discretized without affecting the model inputs and determines the availability of the action to the receiving agent. A '1' enables the analyse action for all of its hosts, whereas a '0' prevents the action.

Once the user agent receives a signal, 'analyse' can be employed, which should result in the detection of a privileged shell. The agent is then permitted to use other actions to neutralise the red's session on the undetected host. Since red can only infiltrate using escalated sessions, and the 'remove' action does not function on these hosts, blue agents over time will learn to re-image a host rather than try to kill a malicious process. This entire approach proved prudent in effectively eliminating any randomness of agent's deploying their actions and also ensured that actions are taken based on direct observations of compromise or through communication. It is important to note that communication itself is constrained, where agents are only allowed to send messages if they detect malicious activity on their monitored hosts. This allows the agents to prevent any unnecessary communications and ensures that messages are relevant.

### 4.3.4 Block Action

By implementing the action masking strategy, the behavior of the agents changed drastically, which allowed the agents to only make informed decisions. However, a significant gap of multiple timesteps is created for attacker to make its moves. Firstly, the operational agent has to recognize the port-scan indicators, then at the next timestep, it has to send an alert of these indicators, to the user agent. Subsequently, the user agent receives the alert after another timestep has elapsed. And lastly, the mitigation actions to remove the red agent from a host is a 2-step process; 'analyse' and 'restore'. In the meantime, the compromise actions of the red agent can be executed immediately after discovering the ports which created an opening for red to intrude into the operational subnet.

Obviously, this necessitated an adjustment of the scenario to allow the defender agents to stop the attacker before its lateral movement. Thus, the 'block' action is implemented which provides a strategic proactive tool for the defender agents. The action is adapted from CybORG's code revision from CAGE challenge 4 and is devised to be used as a single step action. In other words, when an agent employs this action on a particular subnet, it is valid for 1-timestep and at the end of a turn, the block is automatically lifted.

Although, this may not be the most accurate and realistic design of this action, there are a few reasons for this type of functionality. Initially, this action was imagined to be a toggle action, where agents have to use the same action to block and unblock traffic. Another idea was to have two deliberate actions; 'block' and unblock'. Both of these approaches required agents to learn the isolation of networks as well the unblocking of subnets. While experimenting, each of these strategies led to conservative behaviour of the agents, where they learned to maintain the block for longer periods of time. A single-step 'block' aligned well with the communication strategy of the operational agent where once a message is sent, the agent repeatedly blocks the user network until the threat is eliminated.

## 4.4 Game Design for Advanced Configurations

After successful trials of Phase 1 and 2, a natural progression is to assess and evaluate the established game design principles in more complex scenarios. Therefore three new scenarios are devised, where the first two tasks deal with the added uncertainty in the agent detection operation and the last task is concerned with adding an new subnet and an agent.

### 4.4.1 Green Agent

The first scenario in Phase 3 consisted in adding a green agent to the existing scenario. A green agent in CybORG is a normal user that is empowered with performing three actions: 'sleep', spawn a process, and scan an IP for ports. A probability function is assigned for the execution of these actions, where a higher chance is given to the 'sleep' action that essentially does nothing and lower probabilities for the other two actions. This ensures that the use of 'scan' and 'process spawn' actions by the green user is less frequent and may appear as malicious activity to the defender agents. Although only a single green agent is utilized in this scenario, it proved sufficient in CybORG simulator for blue agents to deal with this added complexity.

### 4.4.2 Scan Detection Rate

Throughout the trials, the exploit detection rates are varied to test the development of behaviors and agent strategies. Only the port scan identifiers remained constant which led to the agents devising a communication protocol that helped them in mitigating threats in uncertain conditions. However, this may again not be true in real world situations with the covert nature of malicious actors.

Obviously, a logical approach is to vary the detection rate on port scans and observe the evolution of the schemes of the MARL system. Similar to the visibility of the exploits, detection rates on port scans are lowered to 50% in order to analyse the behavior of the agents. Likewise, both defender agents struggled in this obscured environment where they were only able to deal with threats half of the times. Their communication strategies also proved to be inconclusive due to the fact that agents communicate only after observing some activity on their hosts. To address this adapted strategy, communication is enabled at every timestep regardless of detection.

### 4.4.3 Large Simulated Network

The last task of the experimentation involved the introduction of a new subnet (Enterprise subnet) with four hosts for which an additional agent is unveiled to oversee the added network. To align with the enterprise subnet, the size of the original two subnets is also increased with the addition of two additional hosts: one for each network.

The network configuration can be seen in Figure 4.5. It can be observed that similar to the base scenario established in Section 4.1.2, red agent has a network foothold in the user subnet and its goal is to navigate to the operational server. Only this time, the attacker has to also compromise necessary hosts in the enterprise subnet before it can capture the critical asset. This is an immensely

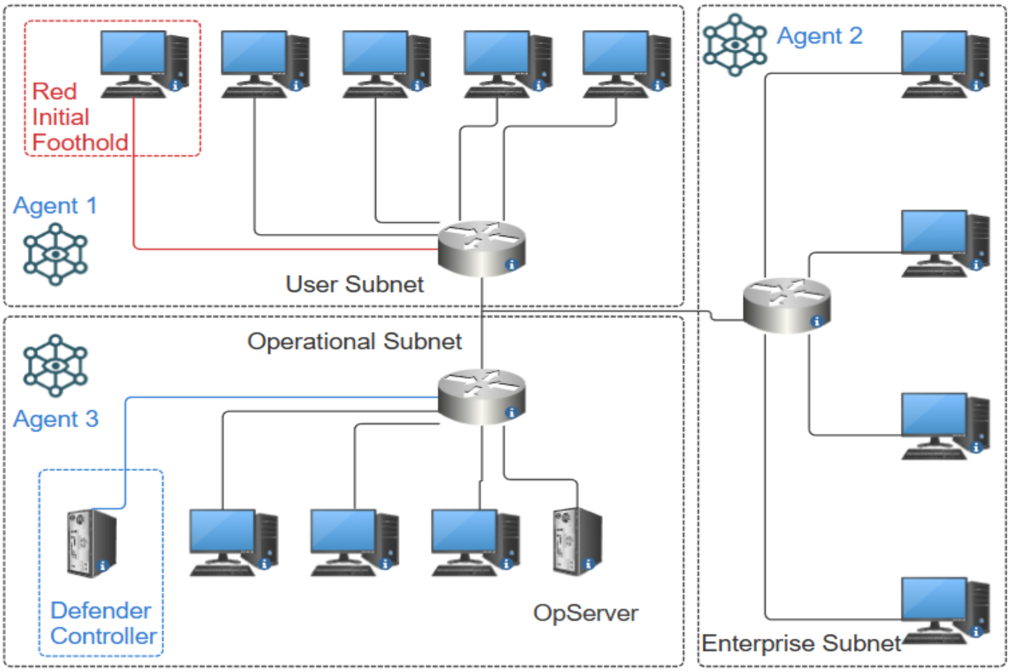


**Figure 4.5:** Large network with three subnets.

complicated problem for MARL agents to solve especially in developing a communication strategy, where now these agents have to devise their mechanisms which involve sending messages to more than 1 agent at a time.

The scenario is set up such that, one of the hosts in the user network has specific configuration to the network interface of the enterprise subnet which allows the red agent to move into this area. From this point, similar to the original scenario, the red agent, after compromising another specific host, moves into the operational zone. All the game design elements designed for Phase 2 remain unchanged for this scenario. However, to maintain a simplistic approach, green agents and the variable detection for port scans are removed.

## 4.5 Methodology Summary

This chapter has delineated the methodology employed to advance the understanding and application of MARL with inter-agent communication in cyber defence scenarios. The research adopts a structured approach to integrate and adapt the DIAL algorithm in CybORG, and aims that with strategic communication between agents, decision-making capabilities can be improved. Additionally, through a phased experimental design, the methodology facilitated the iterative refinement of agent policies to increasingly complex cyber scenarios. The evaluation design and the results of the application of this methodology will be presented in the subsequent chapter.

# 5 Evaluation and Results

This chapter provides the design and process involving the evaluation of the trained MARL agents as well as any key findings of the research. To this end, it is organized to first discuss the criteria for evaluating the experiments which includes the experimental setup, the benchmark, and the evaluation metrics used for the assessment. Next, the results of these experiments including an analysis of the trained policies are outlined. The chapter concludes with a comprehensive discussion of the findings, including the capabilities, limitations and any strategic insights gained from the implementation of communication techniques in cyber defence scenarios.

## 5.1 Evaluation Criteria

As discussed in Chapter 4, this research develops a MARL system, in phases, that enables agents to perform tactical level decision making via communication to counter the advances of an attacker agent. Initially, the system is trained using the cooperative algorithm, QMix, which sets a benchmark for evaluating the learning capabilities of a communicative MARL algorithm, DIAL. Subsequently, the DIAL algorithm is integrated, modified, and adapted to allow agents to convey meaningful messages which hypothetically should lead to better decision making. Additional trials are then conducted to test the ability of the agents in increasingly complex scenarios. The purpose of this evaluation is to assess the effectiveness of DIAL agents in CybORG and to validate the aim of this research that communication can improve agent's performance in ACD tasks.

Recall that unlike traditional MARL algorithms, DIAL incorporates explicit communication among agents, allowing them to share information and make coordinated decisions. DIAL allows agents to learn policies that are contingent on the partial state they observe as well as on the message received from other agents. On the contrary, models that learn independently only account for the state, each agent is permitted to view, and some CLDE architectures process the entire state of the environment to learn joint policies. Thus, DIAL can be interpreted as a hybrid of independent learning and CLDE approaches where it is uniquely equipped to interpret the observations and actions of other agents, albeit in a condensed form.

Training of DIAL agents, in each phase, starts with randomized communication and action policies which are then optimized through the iterative trial and error process. Throughout the training, agents engage in episodes of interactions with the environment, where they not only make decisions on the actions to counteract an attacker but also decide on the communication message to send to other agents. The learning process of DIAL is driven by both the environmental rewards and the effectiveness of communication in improving collective performance.

In the implementation of DIAL, each agent at every timestep generates a communication message, which is real-valued during training and discrete in the execution stage. This message is fed into the network along with partial environment observations, previous timestep action and an agent identifier. The intuition for providing the previous action to the model, lies in the fact that through RNN hidden states, agents will remember all the previous actions they performed in an episode and aid them in choosing the next action. Moreover, the agent identifier is useful because a single network is trained and combining it with the observations and messages allows the policies of each agent to be different. Subsequently, the network processes these inputs to update the agent's policies for both, actions and communication. This mechanism highlights a distinct feature of DIAL where it backpropagates the communication errors directly from one agent to another [25]. This backpropagation influences how communication strategies develop over time, reinforcing messages that lead to successful outcomes and discouraging less effective communication.

The capability of DIAL algorithm within the CybORG environment relies on how well the agents can evolve from their initial state to effectively coordinate their actions through communication. Obviously, the reward function plays a pivotal role in assessing the performance of the DIAL system. Minimizing the accrued penalties over time is a good indicator of the learned defensive strategies. Moreover, a key aspect of this evaluation is observing how the integration of communication influences the overall strategy of the agents compared to baseline models like QMIX, which relies on a central training mechanism without direct agent-to-agent communication. This approach to evaluating the DIAL algorithm aligns with the broader goals of advancing autonomous cyber defence capabilities, offering insights into how different learning architectures can affect the adaptability of MARL systems in practical applications.

### 5.1.1 Benchmarks

Naturally, to test the viability of any MARL system, mechanisms need to be devised that assess the effectiveness of the developed system against a benchmark. Two benchmarks are used for these trials: 1) DIAL - without communication, and 2) QMix. Recall that DIAL is an adaptation of DRQN architecture with additional mechanisms that facilitate multiple agents as well as a communication channel. Thus, DIAL-no comms can essentially be conceptualized as multi-agent DRQN. Although, for DIAL-no comms, a single architecture is used to train the policies of all agents, the strategies are learned and executed independently. In the paper that introduced the DIAL algorithm, results of the experiments on the switch riddle and the MNIST games were measured against this model of independent strategies without any communication [25]. This provides the added motivation of comparing the developed cyber MARL system against fully decentralized policies.

Furthermore, to demonstrate the applicability of MARL methods in cyber defence, Wiebe et al. utilized the QMix algorithm to train agents in CybORG [13]. QMix is inherently a cooperative algorithm, where the policies are learned jointly, but the trained policies operate in a decentralized fashion. Nonetheless, QMix, via its centralized training approach, offers a different avenue for comparison of the overall MARL system.

Both the benchmarks are separately trained for the first two phases of the trials, and only QMix is trained for the last phase of the trials conducted. Due to the consistent under performance of the DIAL-without communications algorithm, it was excluded from the last phase. Moreover, the initial conditions of QMix and no-comms DIAL is set according to Tables 4.1 and 4.4 respectively. Furthermore, the training of the benchmarks is dependent on the game design elements of that specific scenario. However, in phase 2, where action masking is adopted for the DIAL system, it is also implemented for the two benchmarks with a slight modification. As previously discussed, for DIAL implementation, the use of the 'analyse' action is limited and is available only via communication or through a detection of an exploit on a host. For the benchmarks, it is irrational to apply this methodology as there is no mechanism for explicit message relay and therefore, the action will never be utilized by the agents in given conditions. Thus, it is logical to only restrict the 'remove' and 'restore' actions and not the 'analyse' action.

### 5.1.2 Evaluation Metrics

While the original DIAL paper opted for normalized rewards, this research adopts the metrics used by Wiebe et al., with mean return as the primary evaluation metric [13]. Mean return is calculated as the average cumulative reward received per episode across three distinct and independent training iterations, which serves as a good indicator of an agent's performance over time. Training iterations refer to the experiments of a particular scenario that is executed multiple times. This approach does not fully account for variability in the learned policies but ensures that it captures some divergence in the learning process. Furthermore, mean return metric offers a straightforward measure of the system's performance across multiple episodes, providing a clear view of the learning progression. Accordingly, learning curves are plotted at every 10 epochs using this measure.

Once the policies are fully trained, their effectiveness is evaluated by running them through 128 episodes per iteration, capturing both the mean and the standard deviations of the returns. This approach of assessing the trained policy validates the consistency of the learned behaviors. Standard deviation in this context highlights the variability in performance, which is critical for assessing these policies.

In addition to numerical metrics, the learned policies are analyzed during the evaluation phase. This analysis includes observing the strategies adopted by the agents, particularly how they utilize communication in each phase of training. It is essential to understand how communication strategies evolve over time and their impact on the overall effectiveness of the agents in coordinating and responding to cyber threats.

### 5.1.3 Evaluation Process

The evaluation of the DIAL algorithm within the CybORG environment unfolds across three distinct phases. The objective of this multistage evaluation is twofold: 1) to affirm the functionality of a communication enabled MARL algorithm, and 2) to illustrate agent performances across various scenarios through modifications of the communication and action strategies. The first phase sets the foundation that integrates DIAL in a relatively straightforward setup, that facilitates basic communication between agents to address tactical cyber defence tasks. Subsequent phases build on the preliminary results, progressively introducing intricate game design elements aimed at augmenting the real-world applicability of the learning system.

Throughout these phases, training and evaluation are methodically conducted to compare the evolved strategies against established benchmarks. Each phase is designed to isolate and test the influence of specific changes to the system's configuration, from communication protocols to reward structures. Performance metrics are aggregated and averaged across several training iterations that ensures the robustness of the findings. This approach provides a comprehensive assessment of DIAL's efficacy in managing complex network defence operations.

**Phase 1 Trials**: In Phase 1, the research is centered on integrating the DIAL algorithm within the CybORG environment. As detailed in Chapter 4, this phase involves setting up the environment and running the baseline models for evaluating DIAL. Hyperparameters are also tuned here to optimize the system's response, though specifics are elaborated in the Methodology Chapter's, Section 4.2.2. Moreover, efforts are made to adapt CybORG to enable agents to use the communication channel in DIAL, primarily by shaping the rewards, and experimenting with multi-bit messages. Various methods are tested, including different penalties for host captures and illegal actions to encourage more strategic gameplay. For instance, additional penalties were added for the initial compromise of a host with a user shell by the red agent with the penalties doubling upon escalation. However, this did not yield any advantages. Additionally, a 2-bit communication strategy is tested but it also proved insignificant. The simplicity of a 1-bit message aligned well with red agent's sequential compromise pattern, allowing for a condensed but sufficient representation of states of other agent's observations. Furthermore, a reduced exploit detection rate is experimented with and subsequently evaluated which proved beneficial in revealing critical cues for further game design elements.

To evaluate the trained model, mean return along with the training curves are compared and analysed with the benchmarks. The evolution of the agent policies are also monitored across initial, middle and final training stages which provided insights into the effectiveness of communication strategies at different points of the training process. This detailed analysis was important in understanding the practical implications of the tactics across all learning stages as well as in assessing the potential of coordination within the independent MARL framework of CybORG.

**Phase 2 Trials**: This phase introduces more dynamic elements to evaluate the robustness of communication strategies under varying conditions. Specifically, two pivotal changes are implemented: action masking and the addition of a 'block' action. These modifications are aimed at enabling the agent's strategic use of the communication channel and refine agent's tactical decision-making capabilities. The impact of these modifications are significant, with agents demonstrating actual coordination and improved defensive strategies compared to the earlier phase. As with phase 1, the training progress is assessed by comparing learning curves and mean returns against newly established benchmarks for Phase 2.

**Phase 3 Trials**: The third and final phase advanced the environmental design to evaluate the scalability of the developed MARL system. This phase involves the introduction of a benign actor, 'green' agent, where it performs non-malicious actions which added layers of complexity to the agent's operational environment. Additionally, experiments are conducted with varying the port scan detection rates. Lastly, the scenario expanded to include an additional subnet along with a new agent overseeing this network, thus increasing the operational space that the agents are required to defend. This phase did not involve further hyperparameter tuning or changes to the reward structure or communication strategies established in previous phases. Instead, it focused on evaluating how well the strategies developed in earlier trials could adapt to potential ambiguities in the threat landscape.

### 5.1.4 Experimental Setup

The sequence of actions within each timestep is carefully designed, where blue agents are configured to initiate each timestep with their active mitigation actions followed by the red agent's compromise strategies. The blue agent's mitigation actions are generally reactive and is based on the previous moves of the red agent. Typically, until the red agent exploits a host, the defender agent's active actions consist largely of the 'sleep' action, which effectively simulates the attacker's first-move advantage observed in real world settings. Following this, the defender agents are set to passively monitor their respective zones at the end of each turn.

Each trial in the first two phases and the initial tasks of Phase 3 are structured to run for 30 timesteps per episode. The final task of Phase 3 expanded this to 60 timesteps, allowing for more extended engagement due to the introduction of the larger network configuration. These timestep constraints are selected to balance the simulation's complexity with the computational feasibility. In any cyber defencee scenario, the interaction between the red and blue agents could theoretically continue indefinitely. However, for the purpose of this study, limiting the episodes to 30 and 60 timesteps allowed for a focused analysis of the agent's strategies in defending the operational server. Moreover, instead of designing the game with a terminal condition such as the compromise of the operational server, the scenarios are left open-ended to encourage the agents to learn defence strategies without prematurely terminating the episode, ensuring they gain experience in all stages of a cyber battle space.

Training for each trial is extended over 5,000 epochs, with each epoch comprising 128 episodes. This setup resulted in a substantial total of 19.2 million timesteps per trial for the initial phases and 38.4 million timesteps for the larger network, providing a robust dataset for evaluating the learning model's performance and stability over time. Furthermore, each trial configuration is executed across three independent iterations. Average return is calculated to assess the performance variations and to establish confidence in the observed outcomes. Additionally, every 10 epochs, the game scenarios are tested under greedy policies not only to gauge the effectiveness of the learned strategies, but also to use them to plot the learning curves. All performance data are recorded in CSV files for subsequent analysis. Lastly, all simulations are conducted on a Windows 11 system, utilizing only the CPU, with no GPU acceleration. The computational setup included 8 cores, which are essential for multiprocessing, that enables the efficient handling of these simulations.

### 5.1.5 Optimal Score

The optimal score reflects the theoretical best outcome which the learned system can achieve under given conditions. Given the game dynamics, where penalties are accrued for maintaining escalated privileges on a host but not for the initial compromise, the optimal strategy for the blue agents is to remove the red's session corresponding to that host. This will yield a score 0. However, during a 30-timestep game and with a detection rate of 50% at User2 host, there is a chance that an exploit goes undetected at least once or twice. If undetected, red can escalate privileges and commence intelligence gathering on the operational subnet. From that point on, at every timestep, an escalated shell maintained by red on User2 or any subsequent host will result in accumulated penalties for the blue side.

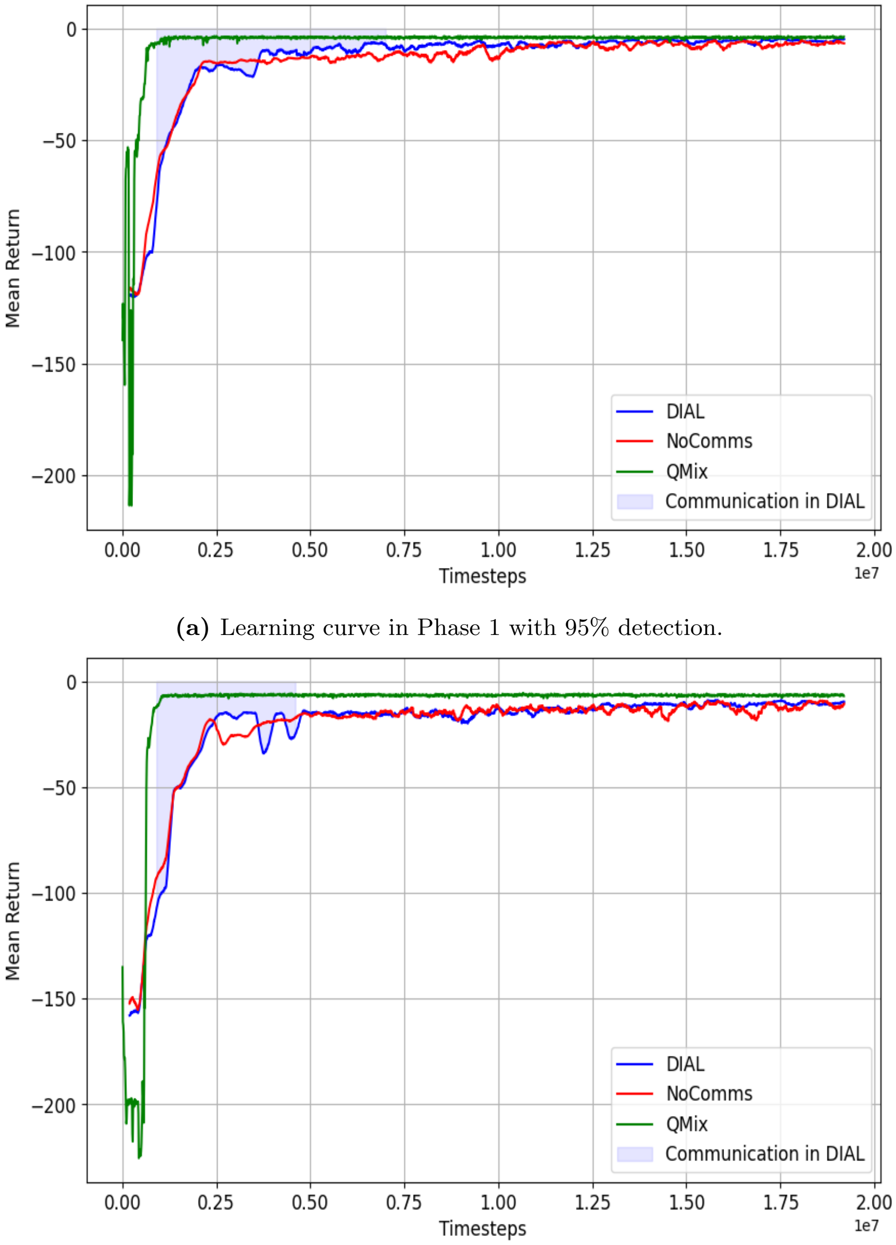
The half chance of missing an exploit necessitates additional strategic actions when the threat eventually becomes apparent. These actions typically include blocking access from the user subnet and restoring the integrity of User2. While these actions are necessary for containment and recovery, they inherently carry penalties which impacts the overall score. Therefore, an optimal score in this setup would be the one where blue agents manage to keep the penalties to a minimum by managing the trade-offs between immediate action costs and the potential for higher penalties if red actions are unchecked.

## 5.2 Results

This section presents the empirical outcomes derived from the experimental trials conducted across the three phases of the study. For each phase, the results are outlined to illustrate the effectiveness of the MARL system with the integration of the DIAL algorithm within the CybORG environment. The results for each phases are presented in their respective subsections, with full sample games for two scenarios and partial sample game for the large network scenario is illustrated in Appendix B, providing a structured overview of how the agent's performance evolved with each experimental setup.

### 5.2.1 Phase 1

The first phase of the trials focuses on assessing the adaptive capabilities of DIAL under varied detection rates and subsequently, compares its performance with the QMix and DIAL-no comms benchmarks. Firstly, models are trained with the default 95% exploit detection rate of CybORG. The learning curves, as illustrated in Figure 5.1a, reveal that QMix marginally outperforms the DIAL algorithm, with DIAL-no comms lagging slightly behind both. This indicates that while DIAL facilitates communication, the high detection rate diminishes the noticeable impact of this feature, as the environment's demands can be met without extensive agent coordination.



**(a)** Learning curve in Phase 1 with 95% detection.

**(b)** Learning curve in Phase 1 with 50% detection.

**Figure 5.1:** Comparative learning curves in Phase 1 with different detection levels.

As detection rates are halved to 50%, the gaps in performance widened as seen in Figure 5.1b. QMix maintains the superiority over both versions of DIAL. This suggests that QMix is better under uncertain conditions, as it effectively handles the reduced information without any reliance on inter-agent communication. The intuition here is that while QMix and DIAL both employ RNN's in their architecture, QMix, due to its centralised learning ability, is able to better equip its agent's to mitigate threats in a timely manner even without detection. As previously stated, in real-world situations, this may not be the best way to deploy agents in cyber defence scenarios, as the threats are far more covert and less frequent.

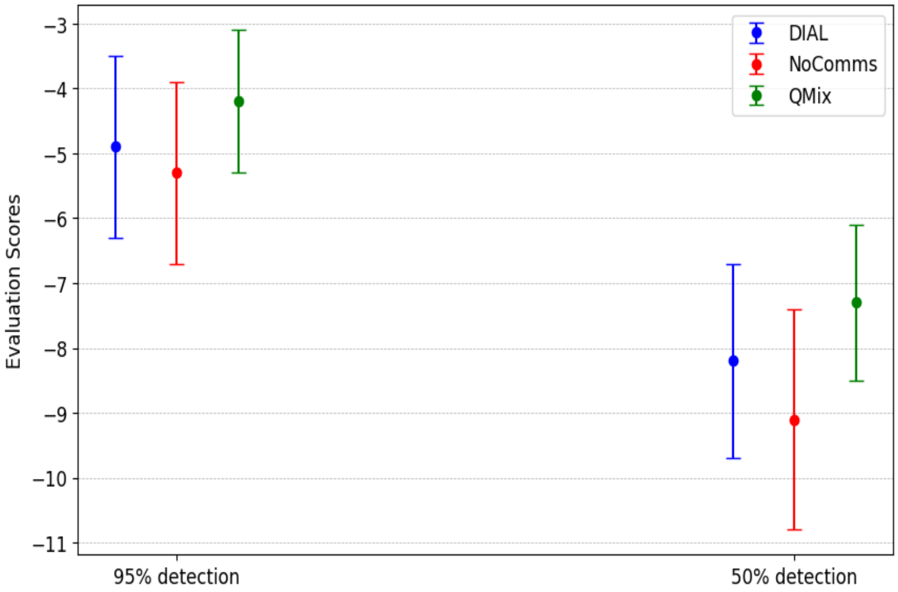


**Figure 5.2:** Evaluation scores of DIAL, DIAL-no comms, and QMix for both trials in Phase 1. Standard deviation is represented by bars.

Figure 5.2 visually details the evaluation scores and their standard deviation for all trials within this phase, illustrating the consistency and fluctuations in each algorithm's performance. Again, it can be observed that during the evaluation of the learned policies, QMix slightly outperformed DIAL, while DIAL-no comms under performed compared to the other two algorithms. Evaluation scores across all phases are listed in Table 5.1.

| Algorithm | 95% detection | 50% detection |
|---|---|---|
| DIAL | $-4.9 \pm 1.4$ | $-8.2 \pm 1.5$ |
| NoComms | $-5.3 \pm 1.4$ | $-9.1 \pm 1.7$ |
| QMIX | $\mathbf{-4.2 \pm 1.1}$ | $\mathbf{-7.3 \pm 1.2}$ |

**Table 5.1:** Evaluation scores across multiple iterations for phase 1, with the standard deviation.

The comparative analysis of learning curves and evaluation scores quantify the performance of each algorithm. However, to understand each algorithm's operational strengths and weaknesses in a simulated cyber defence context, an analysis of the learned policies is warranted. Therefore, the evolved agent policies, as seen in Figure 5.3, across all algorithms and with varied detection rates are investigated. Despite the variances in performance metrics, the learned policies showed similar strategic patterns for all algorithms. Agents are quick to utilize the 'remove' action upon detecting exploits, which is a commonality among all the algorithms and asserts the independent nature of the CybORG environment. However, the nuances emerged more distinctly when exploits went undetected. In all cases agents resorted to the 'restore' action, with QMix timing this action more efficiently after a covert exploit. Both DIAL methods, with and without communication, were slightly slow in utilizing the 'analyse' restore, hence the lower evaluation scores.

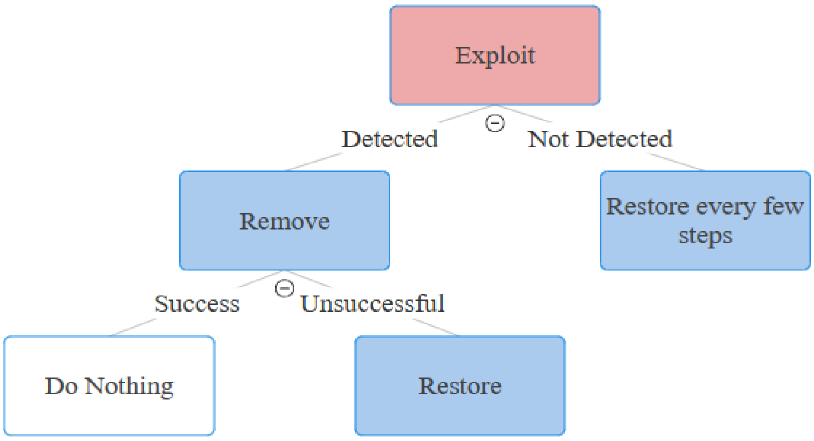


**Figure 5.3:** Trained agent policies in Phase 1 for all scenarios and for all algorithms.

Furthermore, the learned policies of DIAL did not show any substantial communication strategies. There could be many reasons for this behavior but the general perception is that learning additional Q-values for communication is challenging and the RNN architecture forces the agents to resort to independent strategies rather than collaborative. Note that DIAL is originally designed for fully cooperative environments that require communication. The results of the experiments conducted by DIAL authors show that a non communicative algorithm will fail to converge in their fully cooperative environment [25]. In knowing this, it does not help that CybORG can be solved independently with a random use of the 'analyse' action at every few timesteps.

Despite this discouraging behavior of DIAL agents, DIAL displayed signs of its potential in the initial and middle stages of each training iteration. During these stages, meaningful communication is observed as agents utilized the 1-bit message system. The operational subnet agent used the communication channel to alert the agent in the user subnet on the potential threats, such as port scans or exploits. Over time, agents in DIAL veered away from using this strategic communication. Nonetheless, this strategy was instrumental in shaping the modifications implemented in Phase 2.

### 5.2.2 Phase 2

The two benchmarks, QMix and DIAL-no comms, are trained and re-evaluated in this phase with the added game design elements: action masking and the 'block' action. DIAL's adaptability of communication strategies in both of these scenarios are evaluated against the new benchmarks. The learning curves for both the scenarios are illustrated in Figure 5.4 and shows the progression of all three algorithms: DIAL, QMix, and DIAL-no comms.

As evidenced by Figure 5.4a, the inclusion of strategic action masking shows that DIAL slightly outperforms QMix and considerably excels over DIAL-no comms. The addition of the 'block' action significantly boosted DIAL's performance as seen in Figure 5.4b. The corresponding evaluation scores as outlined in Table 5.2 also reveal the same trend of DIAL being better at these tasks compared to the benchmarks. The evaluation scores and standard deviations for this phase are graphically represented in Figure 5.5.

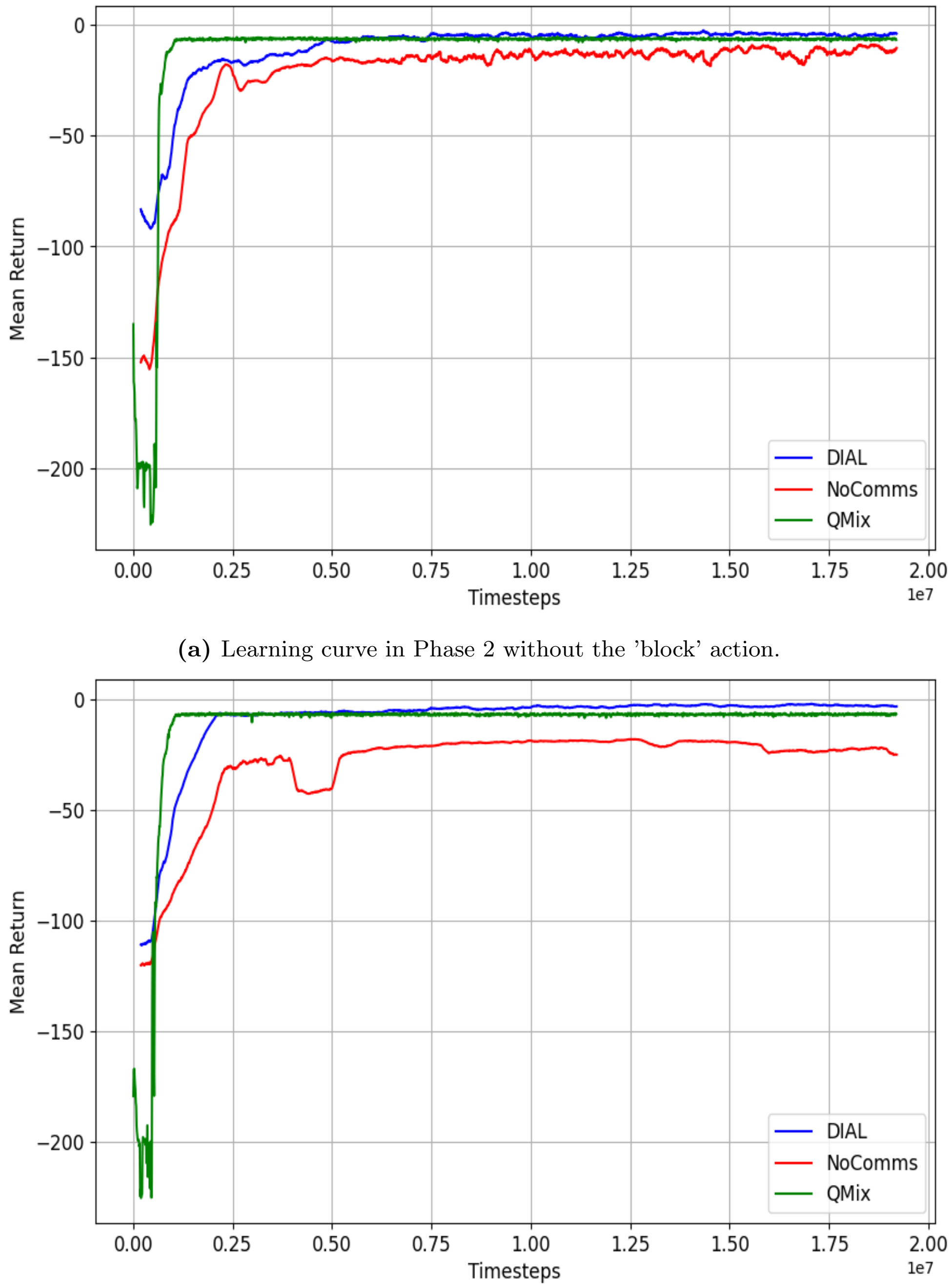
| Algorithm | No Block | With Block |
|---|---|---|
| DIAL | $\mathbf{-6.4 \pm 0.9}$ | $\mathbf{-3.6 \pm 0.8}$ |
| NoComms | $-9.5 \pm 1.5$ | $-16.9 \pm 1.9$ |
| QMIX | $-7.1 \pm 1.3$ | $-7.8 \pm 1.2$ |

**Table 5.2:** Evaluation scores across multiple iterations for phase 2, with the standard deviation.

Analysis of the evolved policies, as evidenced in Figure 5.6, reveal that agents in DIAL effectively utilize the communication channel to coordinate actions like 'analyse' and 'restore' when threats are undetected. This strategic communication is pivotal in scenarios where immediate mitigation is crucial but delayed due to the covert nature of the threat.

For both tasks in this phase, it can be observed from the Figure 5.6a, that similar to phase 1, agents in DIAL swiftly act to remove the attacker from a host upon detection. This is an expected behavior of an optimized policy across all learning phases. However, when a compromise goes undetected and with the imminent attacks on the operational assets, the defender agent in the operational subnet makes a vital decision to alert the user agent of the threat at hand. The operational agent sends the alert in 2 situations: 1) When it observes a port scan, and 2) when it observes an exploit on one of its hosts. In the final policies, the user agent always acted quickly to mitigate the undetected exploit on User2 host and therefore, the operational agent never had to send additional signals after the port scanning. This alert from the operational agent expands the action space of the user agent to include the 'analyse' action, which it promptly uses to disclose a privileged shell of the red agent. It is important to note that although, the analyse action is masked and available only via a communication signal, agents are required to learn this communicative behavior and not a predefined mechanism.

In the task incorporating the block action, the operational agent's immediate response to port scans are twofold: 1) send a signal to the user agent, and 2) isolate the user subnet from where the attack is being initiated as seen if Figure 5.6b. This quick coordination helps bridge the critical gap that existed without the 'block' action, as previously the red agent could move into the operational subnet during this delay. A sample game is provided for the block scenario in



**(a)** Learning curve in Phase 2 without the 'block' action.

**(b)** Learning curve in Phase 2 with the 'block' action.

**Figure 5.4:** Comparative learning curves in Phase 2 with and without 'block' action.

Appendix B.1.

QMix continues to exhibit similar patterns to those observed in Phase 1 for both the tasks, with one difference. Instead of using the 'restore' action, which is masked, it utilizes the 'analyse' action in a similar fashion. This indicates that while effective, its strategies do not diverge significantly with the addition of new

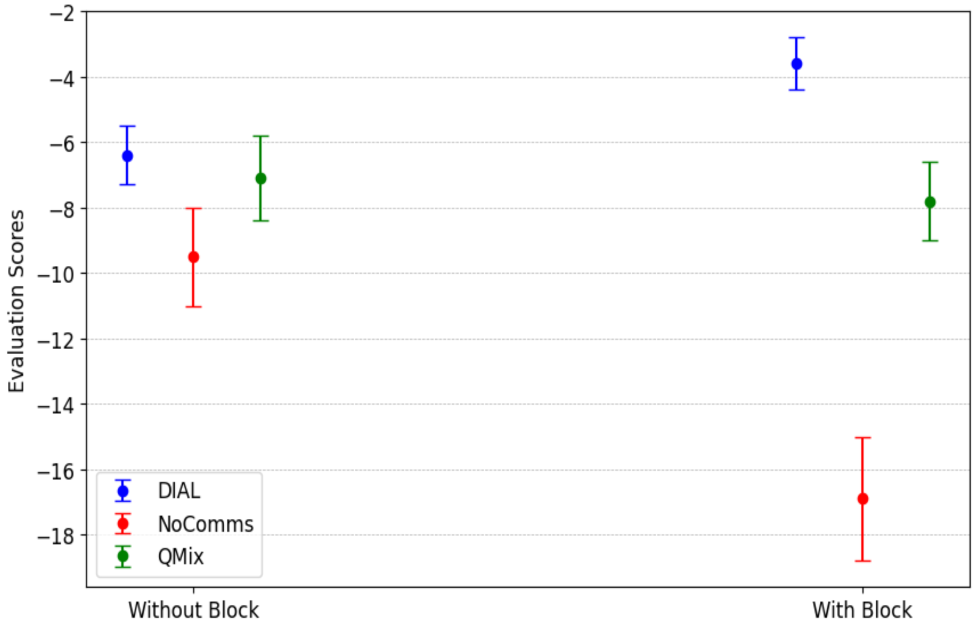
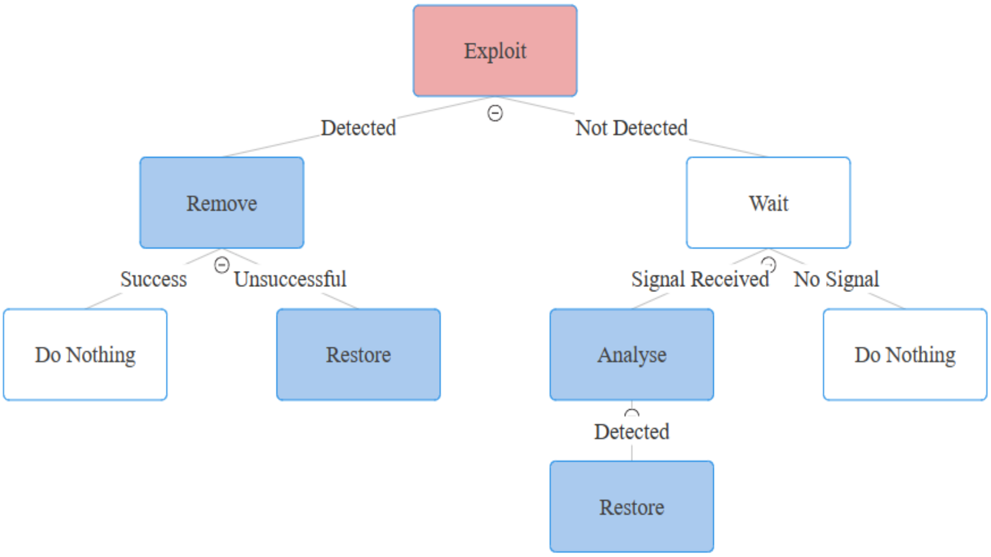
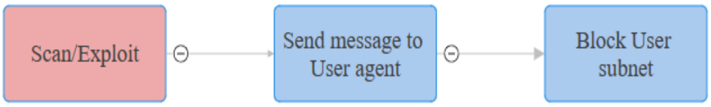


**Figure 5.5:** Evaluation scores of DIAL, DIAL-no comms, and QMix for both trials in Phase 2. Standard deviation is represented by bars.



**(a)** Trained policy for agent 1 across both scenarios.



**(b)** Trained policy for agent 2 with block action.

**Figure 5.6:** Optimized policies of Phase 2 for DIAL.

game elements. It also did not make good use of the ’block’ action which incurs penalties, rather it opted for the use of ’analyse’ action at every few timesteps. ’Analyse’ does not have any penalties if a compromise is detected and has lower penalties than ’block’ if it illegally uses it. Conversely, the no comms strategy follows a similar approach as QMix for the first scenario. However, in the ’block’ action scenario, it adopts an aggressive defence posture, frequently deploying the block action, which although incurs penalties, prevents more significant losses from potential host captures. These results not only illustrate the strategic flexibility of DIAL in a more difficult scenario but also highlights the limitations of non-communicative algorithms, where although the decision making may appear strategic, but inherently they are random.

### 5.2.3 Phase 3

The introduction of a ’green’ agent, simulating benign user behavior that could be misconstrued as malicious by defender agents, marked the first task of this phase. As depicted in Figure 5.7, the learning curves illustrate that while DIAL maintained superior performance over QMix, its effectiveness is somewhat diminished compared to Phase 2. This reduction in performance primarily stems from false positives, where DIAL struggled a little to distinguish between actual threats and benign actions by the green agent. Despite this, DIAL prevented the lateral movement from the user to the operational subnet, with the negative scores largely attributed to the penalties from unnecessary defensive actions. The policy analysis indicates that DIAL’s communication facilitated quick responses, though often triggered by misleading cues from the green agent’s actions as observed in Appendix B.2.

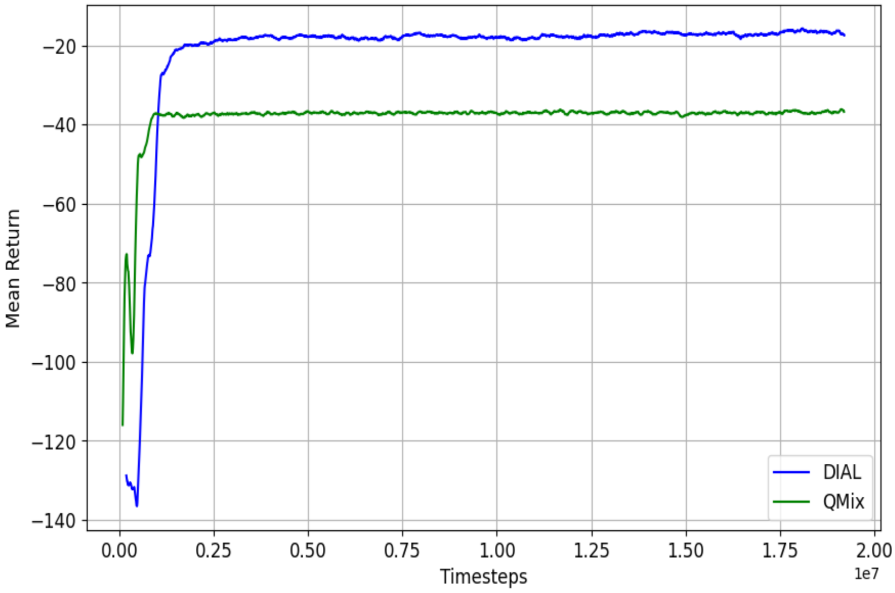


**Figure 5.7:** Learning curves of the different algorithms for the green agent scenario in Phase 3.

The second task experimented with altered detection rates for port scanning and allowed unrestricted communication throughout the episode, removing the green agents for this trial. The scenario kept all the other conditions from Phase 2, including the 'block' action. Interestingly, as illustrated in Figure 5.8, the learning curve shows that DIAL significantly under performed compared to QMix. The graph also shows huge fluctuations in the learning curve, which possibly indicates that the agents policies may have varied throughout the learning phase. After analyzing the policies it became apparent that communication within DIAL depends heavily on reliable threat indicators. The absence of consistent and credible threats led to a breakdown in cooperative strategies, with agents reverting to more independent actions, as detailed in the evaluation scores in Table 5.3. QMix, with its consistent access to the 'analyse' action, better managed the threats before they could laterally move. Another interesting observation is that QMix, does not heavily rely on the port scan detection.

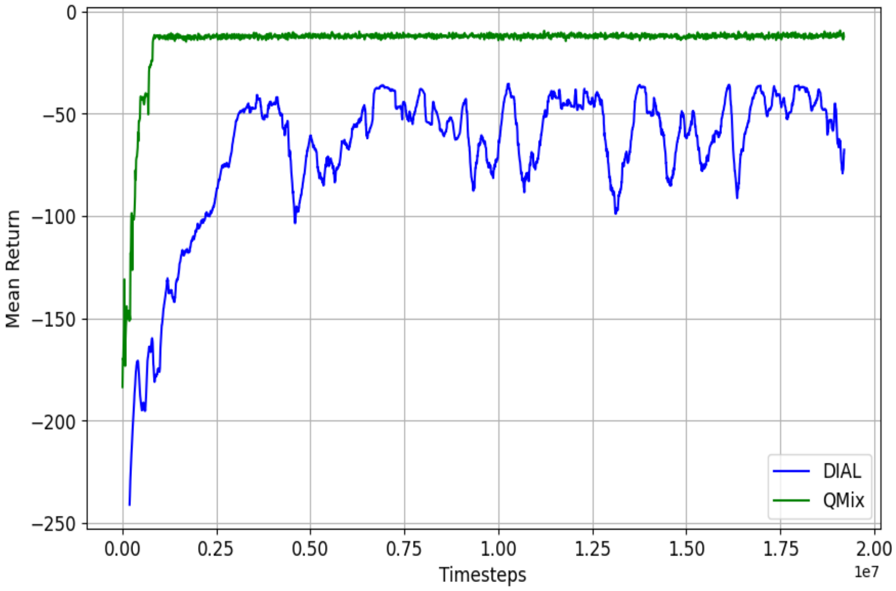


**Figure 5.8:** Learning curve for the varied port scan detection rate in Phase 3.

The final task expanded the network size and introduced an additional agent to manage the increased scope. This setup aimed to test DIAL's performance in a more extensive network environment. Training curves and evaluation scores, presented in Figure 5.9 and Table 5.3 respectively, show that although scores for DIAL are lower compared to the smaller scenario in Phase 2, DIAL significantly outperformed QMix, and successfully adapted to the larger network with the same communication strategies observed in Phase2. The agents not only prevented red's movement to the operational subnet but also managed threats in the intermediary subnet which is illustrated in Appendix B.3.

Overall, Phase 3 demonstrated the potential of DIAL to scale up to more complex network environments while maintaining effective communication among agents. The imperfect detection is obviously a large concern for scenarios that

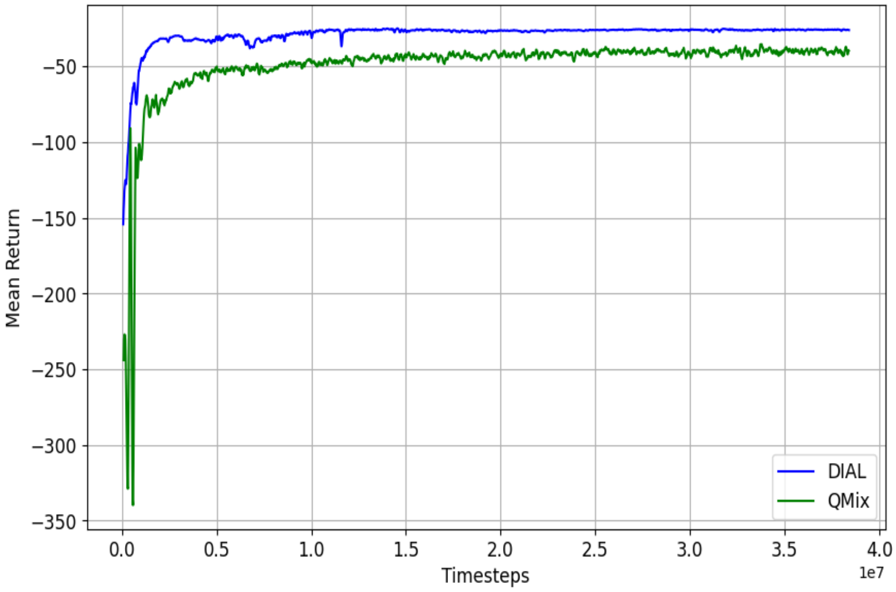


**Figure 5.9:** Learning curve for the large network in Phase 3.

| Algorithm | Green Agent | 50% port scan detection | Large Network |
|---|---|---|---|
| DIAL | $\mathbf{-18.4 \pm 1.8}$ | $-68.7 \pm 22.4$ | $\mathbf{-26.4 \pm 1.5}$ |
| QMIX | $-32.7 \pm 2.3$ | $\mathbf{-9.6 \pm 1.5}$ | $-43.4 \pm 4.6$ |

**Table 5.3:** Evaluation scores across multiple iterations for phase 3, with the standard deviation.

require inter-agent communication to manage threats which will be discussed in detail in the next section. With the exception of one scenario, the results from this phase provide promising indications of DIAL's applicability in realistic cyber defence scenarios.

## 5.3 Discussion

This section delves into some of the intricacies of the developed MARL system within the CybORG environment by reflecting on the challenges and observations throughout the research process. It aims to detail the nuances between the designed communication strategies, the limitations imposed by the simulator, and the implications these have on the broader field of ACD. It will also explore several other key areas including, the role of the 'monitor' action and threat detection, the influence of benign agents in network simulations, and the scalability of the system to larger networks.

### 5.3.1 Communication

In the first phase of the research, the DIAL algorithm within CybORG presented unique challenges in allowing agents to learn communication. Originally, this research envisioned an ideal scenario that would have enabled the communication strategies to be proactive rather than reactive. The strategy would allow agents in the exposed network segments, such as the user subnet in CybORG, to alert other parts of the network. The alert signal would obviously depend on threat indicators within the user subnet. For instance, once the user agent detects a compromise on one of its hosts, it would seek to alert the operational agent of this compromise. In the meantime, while the user agent is busy in dealing with mitigation of an attack, the operational agent would employ a defensive posture, which would include CybORG's 'misinform' action and the 'block' action. The 'misinform' action, which is not utilized in this research, allows any agent to create mock processes on hosts, that appear real to the attacker agent and essentially is designed to delay any exploits initiated by Red. This proactive communication was conceptualized to simulate cyber defence dynamics where early warnings from compromised segments could trigger preemptive defences across the network. Unfortunately, limitations in DIAL algorithm's adaptation to CybORG as well as CybORG's structure hindered the realization of such dynamics.

In DIAL, communication is an indirect aspect of learning, where the formulated messages are not directly tied to immediate rewards. Instead, the value of a message is learned through its impact on subsequent environment interactions and associated rewards which led to fluid communication patterns. This indirect feedback loop made it challenging to directly quantify the impact of a specific message on overall strategies. Additionally, DIAL is designed to encode messages as part of a broader decision-making process. For example, in the Multi-Step MNIST game from the experiments that presented DIAL, communication plays a pivotal and sequential role, where agents have to devise a protocol to identify other agent's MNIST digit as illustrated in Figure 5.10 [25]. In contrast, CybORG's scenarios did not inherently support such complex communication dynamics.

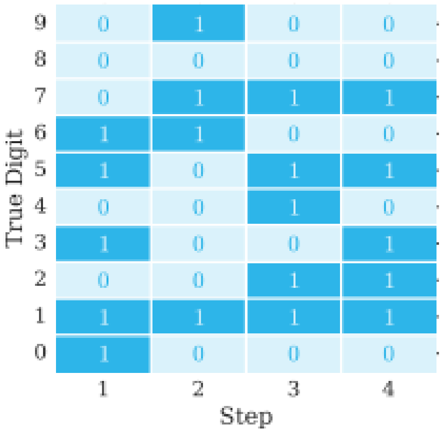


**Figure 5.10:** Protocol of Multi-Step MNIST game [25].

Furthermore, a misalignment is noted in the behaviors of each agent. Due to the sequential patterns of the attacker, the defender agent in the user subnet performed majority of the mitigation actions and the operational agent remained inactive until signs of a compromise is detected. This highlighted a fundamental mismatch between the intended use of DIAL and the intrinsic design of CybORG as an environment geared more towards independent agent operation.

Nonetheless, this research overcame the above challenges. Firstly, while the initial objective of the research to model the proactive communication mechanism is unsuccessful, the game design elements that are eventually adopted are effective within the confines of the simulation, mirroring the conventional incidence response tactics. Secondly, DIAL's encoding strategy is adapted to enable the operational agent to relay a simple signal which proved sufficient for a coordinated autonomous defence. Lastly, the importance of the operational agent is expanded, where now along with its environment actions, it has to utilize the communication mechanism to inform other agent's of imminent attack on its hosts.

### 5.3.2 Monitor action and threat detection

The reliance of the 'monitor' action to trigger any defensive response highlights a critical aspect of cyber defence: the detection of threats. The simulation's setup, where the network is monitored passively without necessitating explicit agent action, is akin to a host-based IDS. This setup simplifies the decision-making process by allowing agents to continuously receive information from this "IDS". Therefore, any failures in threat detection is a failure of the IDS in capturing threat data. However, this may limit realistic training conditions and poses an important question of whether a more active learning approach of this action may have induced more nuanced strategies.

Addressing threat detection raises another important question: how can systems be designed to more confidently detect threats? While it is not the focus of this research to devise better detection mechanisms, it is clear that in applying RL for ACD, the focus often rests on responding to observed threats rather than the detection methods. However, without high-confidence threat detection, the effectiveness of MARL systems in a defensive role might be compromised. Furthermore, the performance degradation observed when the port scan detection rates were lowered in the second task of Phase 3 indicates the sensitivity of the system to the reliability of threat detection. This scenario underscores the delicate balance between detection accuracy and response effectiveness, emphasizing the need for robust detection mechanisms as a foundation for ACD.

Another approach that is extensively discussed in literature is to feed the agents with host and network logs which would allow them to learn to detect threats from these logs. However, such a learning system would require realistic network traffic to be simulated in ACO environments. This is obviously a big limiting factor of today's simulated environments. Furthermore, with this type of approach, the complexity in the agent's learning ability also increases to not only learn the detection mechanisms from network logs but also to perform mitigation actions from these detections.

Moreover, the use of the 'analyse' action also revealed some challenges. Initially, agents did not benefit in the use of this action, primarily due to the high exploit detection rate but also due to no contingencies on the 'remove' and 'restore' actions. For instance, when a threat is detected, agents learned to use the 'remove' action, and when a threat is not detected, agents resorted to using the 'restore' action every few timesteps as a more proactive approach. The 'analyse' action proved beneficial only when the 'restore' action is masked to work when a threat is detected. Although this strategy is useful in balancing the use of the actions, the defender agents shifted their policies to use the 'analyse' action more often. This suggests that while 'analyse' is critical under certain conditions, its use needs careful consideration to avoid excessive resource expenditure. The question then becomes whether to employ the 'analyse' action sporadically to ensure higher threat detection certainty or to use it in an efficient manner such as a relayed message signal from another agent within the network.

### 5.3.3 Green agent

Green agents in the CybORG simulator offers some valuable insights into the complexities of network defence in real-world settings. In a typical enterprise network, normal users generate substantial network traffic, which can mask malicious activities. This is obviously a challenge for the network analysts in differentiating between malicious and benign activities. The incorporation of green agents in CybORG is aimed at replicating this aspect by introducing actions, such as process spawning or port scanning.

The presence of a single green agent significantly increased the complexity of decision-making for the defender agents, leading to a notable accumulation of penalties. While these penalties are not necessarily indicative of actual host compromises, they represent the costs, both in terms of time and resources, associated with addressing false positives. Such costs are a realistic aspect of cyber operations, where not all security alerts correspond to genuine threats, yet they require investigation and consume valuable resources.

Despite this, DIAL agents demonstrate an ability to handle the added complexity in a manner that may reflect realistic defence of cyber systems. For instance, when either green or red agents conducted port scans on a host in the operational subnet, the defender agents utilized the communication channel. Agent in the operational subnet alerts the user agent, leading to a defensive measure of analyzing all hosts in subsequent timesteps. Once the 'analyse' action is used, the 'restore' action is triggered if a threat is detected, otherwise the agent opted for the 'sleep' action.

### 5.3.4 Large Network

The broader objective of this research is to expand the developed methodologies to larger, more complex environments. To explore scalability, the simulation introduced an additional agent and a new subnet, increasing from two hosts per subnet in the base scenario to four. This increase resulted in each agent managing a larger action space, growing from 8 to 15 possible actions. To manage this complexity, the strategy of action masking proved valuable. By selectively narrowing the actions available based on a host compromise, agents could focus on pertinent actions related to the compromised host. This selective focus helps in maintaining efficiency without overwhelming the agents with too many choices at each step.

Interestingly, the expanded scenario with 60 timesteps revealed a notable trend. Despite the fact that the red agent succeeded in compromising the operational subnet early in the training, the evolved strategies as seen in Appendix B.3 prevented the attacker from advancing beyond the intermediary subnet. The comparison of results between the base scenario and the expanded network showed that while the specific scores might differ, the underlying policies learned by the agents remained consistent. Although definitive conclusions about scalability to even larger networks remain untested, the success in a 3-agent environment suggests that the agents are capable of scaling their strategies to larger networks, obviously with careful considerations of the communication strategies.

## 5.4 Evaluation Summary

This evaluation chapter has presented the outcomes of integrating and testing the DIAL framework within the CybORG simulated environment through various experimental phases. The results validate and demonstrate the algorithm's capacity in enabling agents to formulate both the communication as well as the defensive strategies. The performance of the DIAL algorithm, compared against QMix and the non-communicative variant, points to the potential benefits of structured communication in multi-agent settings for ACD. Although the DIAL algorithm outperformed in scenarios requiring active communication for coordination, it also revealed limitations in scenarios with decreased detection capabilities, highlighting the importance of precise threat detection.

Notably, the addition of more complex game elements, such as the introduction of benign green agents and the expansion of network, provided valuable insights into the scalability of the learned behaviors. These experiments showed that while the agents are capable of adapting to complex environments, the performance is influenced by the realism of the threat models employed.

# 6 Conclusion

This Chapter concludes this research, first, by highlighting the contributions made towards the broader scope of ACD. It will also suggest future research directions based on the observations and insights gained during the methodology and the evaluation processes of this research.

## 6.1 Contributions

To the best of our knowledge, this research on the application of multi-agent communication frameworks for ACD simulations marks an advancement in the field. A major contribution of this work is the successful implementation and validation of a communicative MARL algorithm within an ACD environment, demonstrating its practical utility in cyber defence scenarios. This adaptation facilitated strategic decision-making by enabling agents to communicate. It allowed agents to utilize their actions in an informed manner rather than randomly.

Another key contribution is the identification of the limitations that exist in current cyber defence simulations. The research found critical gaps in threat mitigation capabilities, particularly under conditions of imperfect detection rates. It also identified challenges posed by the abstraction levels in CybORG and similar environments. These abstractions, which simplify elements like agent observations, actions, operating system complexities, and network intricacies, may not accurately reflect the operational challenges faced in real-world cyber defence settings. Furthermore, the findings suggest that the configuration of agents, where each agent is responsible for a specific subnet, may not optimally leverage the potential for collaborative threat mitigation.

## 6.2 Future Work

The study presented in this research highlights the potential of MARL based algorithms that facilitate inter-agent communication in cyber defence scenarios. While the aim of this research is successfully achieved, a few complexities that were originally envisioned remained untested within the context of the DIAL algorithm as follows:

- **Multiple Red Agents**: The setup used throughout the research involved a single adversarial agent, which allowed for a simplified threat dynamics. The theory of introducing multiple red agents would enable a simulation where multiple threats arise simultaneously. This scenario would be challenging for the defender agents where now they would have to prioritize and manage multiple threats. This setup would test the robustness of the communication strategies developed and the expectation is that, with some adjustments, the coordination tactics would not diverge from the ones that are presented in this work.

- **Multiple Green Agents**: To realistically mirror the noisy network traffic found in enterprise network settings, this research hoped to incorporate multiple green agents and evaluate in comparison to the single green agent scenario of Phase 3.

- **Individual Host Responsibility**: Shifting from subnet-based to host-based responsibilities for each agent naturally benefits with a more refined action space where now agents would be responsible for a small number of actions. The change would potentially increase the inter-agent coordination due to a more distributed task structure. However, this approach would require reevaluation of the communication mechanism, possibly extending the 1-bit communication to multi-bit to handle the need for detailed information sharing.

Additionally, this research discussed some limitation in Section 6.1 and therefore, to overcome them, a few topics in the broader context of cyber defence simulators and MARL are presented as follows:

- **Realism in Simulation**: Training AI agents directly on real networks is impractical. Simulators must therefore replicate the complexities of these environments accurately. CybORG simplifies certain elements to streamline its cyber games between red and blue agents. Consequently, simulators must evolve to encapsulate the reality of network dynamics more comprehensively. Improving simulators to encompass a broader scope of network operations, attack methods, and defensive mechanisms will create a more authentic environment to test and evaluate communicative MARL strategies.

- **Revised MARL Structures**: Considering the limitations observed with the 'single agent responsible for an entire subnet' structure, a more integrated MARL strategy could be beneficial. This might involve heterogeneous roles among agents, such as specialized monitoring agents and action-specific agents (e.g., remove, restore). Such a configuration could enable for more dynamic cooperation. For example agents monitoring a network have to relay information to agents who are responsible for the mitigation actions.

- **Exploration of other MARL architectures**: So far only value-based MARL algorithms have been explored in ACD, that includes the communication mechanism of DIAL which essentially is a Q-value network. Investigating other frameworks such as policy-based methods could offer new avenues for learning communication and action strategies in a MARL setting.

## 6.3 Closing Remarks

This research successfully validated the aim presented in Chapter 1, to demonstrate the importance of communication between blue agents by showing that relaying key information allows these agents to stop a malicious actor from compromising hosts across subnets. In doing so, DIAL algorithm is first integrated in the CybORG simulation environment. Then, DIAL is adapted in CybORG and enables agents to send a 1-bit binary message as an abstraction of its observations. Subsequently, CybORG is modified to utilize the action masking strategy, particularly with the 'analyse' action that enables agents to make informed decisions on its mitigation actions. CybORG is also revised to include a 'block' action that significantly changed the behavior of DIAL agents to stop the lateral movement. Following this, the game design is evaluated in varied conditions, that included green agents of CybORG, imperfect detection and a larger network.

Training graphs and evaluation scores are then compared with benchmarks across all the tasks. Agent policies are also analyzed which provided valuable insights into the adaptability and scalability of the evolved communication strategies. The training setups, specifically in the latter phases, tested the agent's abilities to cope with increased complexity that widened the understanding of some of the limitations imposed by the current architectural settings within simulated environments. In particular, threat detection with more confidence is identified as a critical factor for any scenario requiring meaningful communication between agents. Additionally, the MARL structure implemented in CybORG, where each agent manages an individual subnet, proved challenging for agents to learn to coordinate their actions.

Nonetheless, it is evident that MARL with communication offers an invaluable tool in the development of blue agents for real world cyber threat mitigation. This research provides a path for future work to address the limitations outlined, while also adapting with the knowledge and understanding from the evaluation presented in this work.

# References

[1] I. H. Sarker, A. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *Journal of Big data*, vol. 7, pp. 1–29, 2020. [Online]. Available: https://link.springer.com/article/10.1186/s40537-020-00318-5

[2] H. Orman, "The morris worm: A fifteen-year perspective," *IEEE Security & Privacy*, vol. 1, no. 5, pp. 35–43, 2003. [Online]. Available: https://ieeexplore.ieee.org/document/1236233/

[3] P. Institute, "Cost of a data breach report," 2023. [Online]. Available: https://www.ibm.com/reports/data-breach

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2015. [Online]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/26819042/

[6] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *arXiv preprint arXiv:1509.01549*, 2015. [Online]. Available: https://arxiv.org/abs/1509.01549

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8103164/

[8] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2021. [Online]. Available: https://arxiv.org/abs/1906.05799

[9] N. A. Grupen, D. D. Lee, and B. Selman, "Multi-agent curricula and emergent implicit signaling," *arXiv preprint arXiv:2106.11156*, 2021. [Online]. Available: https://arxiv.org/abs/2106.11156

[10] S. Vyas, J. Hannay, A. Bolton, and P. P. Burnap, "Automated cyber defence: A review," *arXiv preprint arXiv:2303.04926*, 2023. [Online]. Available: https://arxiv.org/abs/2303.04926

[11] A. OroojlooyJadid and D. Hajinezhad, "A Review of Cooperative Multi-Agent Deep Reinforcement Learning," 2019. [Online]. Available: https://arxiv.org/abs/1908.03963

[12] R. Bhosale, S. Mahajan, and P. Kulkarni, “Cooperative machine learning for intrusion detection system,” *International Journal of Scientific and Engineering Research*, vol. 5, no. 1, pp. 1780–1785, 2014. [Online]. Available: https://www.ijser.org/researchpaper/Cooperative-Machine-Learning-For-Intrusion-Detection-System.pdf

[13] J. Wiebe, R. A. Mallah, and L. Li, “Learning Cyber Defence Tactics from Scratch with Multi-Agent Reinforcement Learning,” 2023. [Online]. Available: https://arxiv.org/abs/2310.05939

[14] PyMARL 2, 2021, accessed: 2023-09-05. [Online]. Available: https://github.com/hijkzzz/pymarl2

[15] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, “Cyborg: A gym for the development of autonomous cyber agents,” *arXiv preprint arXiv:2108.09118*, 2021. [Online]. Available: https://arxiv.org/abs/2108.09118

[16] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://hal.science/hal-04206682/document

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf

[18] M. Hausknecht and P. Stone, “Deep recurrent q-learning for partially observable mdps,” in *2015 aaai fall symposium series*, 2015. [Online]. Available: https://cdn.aaai.org/ocs/11673/11673-51288-1-PB.pdf

[19] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, “Multi-Agent Reinforcement Learning: A Review of Challenges and Applications,” *Applied Sciences*, vol. 11, no. 11, p. 4948, May 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/4948

[20] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, “Multiagent cooperation and competition with deep reinforcement learning,” *PloS one*, vol. 12, no. 4, p. e0172395, 2017. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0172395

[21] L. Busoniu, R. Babuska, and B. De Schutter, “A Comprehensive Survey of Multiagent Reinforcement Learning,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, Mar. 2008. [Online]. Available: https://ieeexplore.ieee.org/document/4445757/

[22] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, “Monotonic value function factorisation for deep

multi-agent reinforcement learning," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 7234–7284, 2020. [Online]. Available: https://arxiv.org/pdf/1803.11485.pdf

[23] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11794

[24] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017. [Online]. Available: https://arxiv.org/pdf/1706.05296.pdf

[25] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to Communicate with Deep Multi-Agent Reinforcement Learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf

[26] S. Sukhbaatar, a. szlam, and R. Fergus, "Learning Multiagent Communication with Backpropagation," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/55b1927fdafef39c48e5b73b5d61ea60-Paper.pdf

[27] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents," 2018. [Online]. Available: http://proceedings.mlr.press/v80/zhang18n.html

[28] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games," 2017. [Online]. Available: https://arxiv.org/abs/1703.10069

[29] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, "TarMAC: Targeted Multi-Agent Communication," 2018. [Online]. Available: https://arxiv.org/abs/1810.11187

[30] A. Singh, T. Jain, and S. Sukhbaatar, "Learning when to Communicate at Scale in Multiagent Cooperative and Competitive Tasks," 2018. [Online]. Available: https://arxiv.org/abs/1812.09755

[31] G. Palmer, C. Parry, D. J. Harrold, and C. Willis, "Deep reinforcement learning for autonomous cyber operations: A survey," *arXiv preprint arXiv:2310.07745*, 2023. [Online]. Available: https://arxiv.org/pdf/2310.07745

[32] N. Wagner, C. Ş. Şahin, M. Winterrose, J. Riordan, J. Pena, D. Hanson, and W. W. Streilein, "Towards automated cyber decision support: A case study on network segmentation for security," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7849908

[33] L. Li, R. Fayad, and A. Taylor, "Cygil: A cyber gym for training autonomous agents over emulated network systems," *arXiv preprint arXiv:2109.03331*, 2021. [Online]. Available: https://arxiv.org/abs/2009.08120

[34] W. Blum, "Gamifying machine learning for stronger security and ai models," 2021. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2021/04/08/gamifying-machine-learning-for-stronger-security-and-ai-models/

[35] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and self-play," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9. [Online]. Available: https://arxiv.org/abs/2009.08120

[36] A. Molina-Markham, C. Miniter, B. Powell, and A. Ridley, "Network environment design for autonomous cyberdefense," *arXiv preprint arXiv:2103.07583*, 2021. [Online]. Available: https://arxiv.org/pdf/2103.07583

[37] TTCP CAGE Challenge, 2021, accessed: 2023-09-05. [Online]. Available: https://github.com/cage-challenge

[38] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016. [Online]. Available: https://arxiv.org/pdf/1610.03295

[39] Z. Utic and K. Ramachandran, "A survey of reinforcement learning in intrusion detection," in *2022 1st International Conference on AI in Cybersecurity (ICAIC)*. IEEE, 2022, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/9897058

[40] X. Guo, J. Ren, J. Zheng, J. Liao, C. Sun, H. Zhu, T. Song, S. Wang, and W. Wang, "Automated penetration testing with fine-grained control through deep reinforcement learning," *Journal of Communications and Information Networks*, vol. 8, no. 3, pp. 212–220, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10272349

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[42] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE*

*Symposium on Security and Privacy.* IEEE, 2002, pp. 273–284. [Online]. Available: https://ieeexplore.ieee.org/document/1004377

[43] Y. Sun, W. Xiong, Z. Yao, K. Moniz, and A. Zahir, “Network defense strategy selection with reinforcement learning and pareto optimization,” *Applied Sciences*, vol. 7, no. 11, p. 1138, 2017. [Online]. Available: https://pdfs.semanticscholar.org/4f3c/53bba5acfa7507c4c487c71eaf74771dc382.pdf

[44] M. Kiely, D. Bowman, M. Standen, and C. Moir, “On autonomous agents in a cyber defence environment,” *arXiv preprint arXiv:2309.07388*, 2023. [Online]. Available: https://arxiv.org/pdf/2309.07388

[45] A. Dutta, S. Chatterjee, A. Bhattacharya, and M. Halappanavar, “Deep reinforcement learning for cyber system defense under dynamic adversarial uncertainties,” *arXiv preprint arXiv:2302.01595*, 2023. [Online]. Available: https://arxiv.org/pdf/2302.01595

[46] J. Nyberg and P. Johnson, “Training automated defense strategies using graph-based cyber attack simulations,” *arXiv preprint arXiv:2304.11084*, 2023. [Online]. Available: https://arxiv.org/pdf/2304.11084

[47] S. Vanneste, G. de Borrekens, S. Bosmans, A. Vanneste, K. Mets, S. Mercelis, S. Latré, and P. Hellinckx, “Learning to communicate with reinforcement learning for an adaptive traffic control system,” in *Advances on P2P, Parallel, Grid, Cloud and Internet Computing: Proceedings of the 16th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2021).* Springer, 2022, pp. 207–216. [Online]. Available: https://arxiv.org/pdf/2110.15779

[48] M. Panfili, A. Giuseppi, A. Fiaschetti, H. B. Al-Jibreen, A. Pietrabissa, and F. D. Priscoli, “A game-theoretical approach to cyber-security of critical infrastructures based on multi-agent reinforcement learning,” in *2018 26th Mediterranean Conference on Control and Automation (MED).* IEEE, 2018, pp. 460–465. [Online]. Available: https://ieeexplore.ieee.org/document/8442695

[49] M. Basnet and M. H. Ali, “Deep reinforcement learning-driven mitigation of adverse effects of cyber-attacks on electric vehicle charging station,” *Energies*, vol. 16, no. 21, p. 7296, 2023. [Online]. Available: https://www.mdpi.com/1996-1073/16/21/7296

[50] G. Shi and G. He, “Collaborative multi-agent reinforcement learning for intrusion detection,” in *2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*, 2021, pp. 245–249. [Online]. Available: https://ieeexplore.ieee.org/document/9660402

[51] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10295

[52] G. McDonald *et al.*, “Competitive reinforcement learning for autonomous cyber operations,” 2023. [Online]. Available: https://espace.rmc.ca/jspui/bitstream/11264/1227/1/Competitive_RL_for_ACO.pdf

[53] G. W. Brown, “Iterative solution of games by fictitious play,” *Act. Anal. Prod Allocation*, vol. 13, no. 1, p. 374, 1951. [Online]. Available: https://www.math.ucla.edu/~tom/stat596/fictitious.pdf

[54] T. Yu and H. Zhu, “Hyper-parameter optimization: A review of algorithms and applications,” *arXiv preprint arXiv:2003.05689*, 2020. [Online]. Available: https://arxiv.org/pdf/2003.05689.pdf

[55] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Surveying port scans and their detection methodologies,” *The Computer Journal*, vol. 54, no. 10, pp. 1565–1581, 2011. [Online]. Available: https://www.researchgate.net/profile/Dhruba-K-Bhattacharyya/publication/262321316_Surveying_Port_Scans_and_Their_Detection_Methodologies/links/56e2541608aebc9edb19d3ed/Surveying-Port-Scans-and-Their-Detection-Methodologies.pdf

[56] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, “StarCraft II: A New Challenge for Reinforcement Learning,” 2017. [Online]. Available: https://arxiv.org/abs/1708.04782

[57] S. Huang and S. Ontañón, “A closer look at invalid action masking in policy gradient algorithms,” *arXiv preprint arXiv:2006.14171*, 2020. [Online]. Available: https://arxiv.org/pdf/2006.14171.pdf

# A DIAL Algorithm

Following is the algorithm for DIAL as detailed in [25]:

**Algorithm 4** Differentiable Inter-Agent Learning (DIAL)

```
Initialise $\theta_1$ and $\theta_1^-$
for each episode $e$ do
    $s_1$ = initial state, $t = 0$, $h_0^a = 0$ for each agent $a$
    while $s_t \neq$ terminal and $t < T$ do
        $t = t + 1$
        for each agent $a$ do
            Get messages $\hat{m}_{t-1}^{a'}$ of previous time-steps from agents $m'$ and evaluate C-Net:
            $Q(\cdot), m_t^a = \text{C-Net}\left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a; \theta_i\right)$
            With probability $\epsilon$ pick random $u_t^a$, else $u_t^a = \max_a Q\left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i\right)$
            Set message $\hat{m}_t^a = \text{DRU}(m)$, where $\text{DRU(m)} = \begin{cases} \text{Logistic}(\mathcal{N}(m_t^a, \sigma)), & \text{if training, else} \\ \mathbb{1}\{m_t^a > 0\} \end{cases}$
        end for
        Get reward $r_t$ and next state $s_{t+1}$
    end while
    Reset gradients $\nabla\theta = 0$
    for $t = T$ to 1, $-1$ do
        for each agent $a$ do
            $y_t^a =$ if $s_t$ terminal then
              $r_t$
            else
              $r_t + \gamma \max_u Q\left(o_{t+1}^a, \hat{m}_t^{a'}, h_t^a, u_t^a, a, u; \theta_{-i}\right)$
            Accumulate gradients for action:
            $\Delta Q_t^a = y_t^a - Q\left(o_j^a, h_{t-1}^a, \hat{m}_{t-1}^{a'}, u_{t-1}^a, a, u_t^a; \theta_i\right)$
            $\nabla\theta = \nabla\theta + \frac{\partial}{\partial\theta}(\Delta Q_t^a)^2$
            Update gradient chain for differentiable communication:
            $\mu_j^a = \mathbb{1}\{t < T - 1\} \sum_{m' \neq m} \frac{\partial}{\partial \hat{m}_t^a} \left(\Delta Q_{t+1}^{a'}\right)^2 + \mu_{t+1}^{a'} \frac{\partial \hat{m}_{t+1}^{a'}}{\partial \hat{m}_t^a}$
            Accumulate gradients for differentiable communication:
            $\nabla\theta = \nabla\theta + \mu_t^a \frac{\partial}{\partial m_t^a} \text{DRU}(m_t^a) \frac{\partial m_t^a}{\partial\theta}$
        end for
    end for
    $\theta_{i+1} = \theta_i + \alpha\nabla\theta$
    Every $C$ steps reset $\theta_i^- = \theta_i$
end for
```

# B Sample Games

## B.1 Game play for policy analysis in Phase 2 - Scenario 2: 'Block'

### Turn 0

Red Action None

Defender Actions None

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 1

Red Action DiscoverRemoteSystems UserSubnet

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 2

Red Action DiscoverNetworkServices User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Scan | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 3

Red Action DiscoverNetworkServices User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Scan | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 4

Red Action ExploitRemoteService User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Exploit | User shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 5

Red Action ExploitRemoteService User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User2 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 6**

Red Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 7**

Red Action PrivilegeEscalate User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 8**

Red Action PrivilegeEscalate User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 9

Red Action ExploitRemoteService User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Exploit | User shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 10

Red Action ExploitRemoteService User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User2 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 11

Red Action PrivilegeEscalate User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 12**

Red Action PrivilegeEscalate User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 13**

Red Action ExploitRemoteService User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Exploit | User shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 14**

Red Action ExploitRemoteService User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User2 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 15

Red Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 16

Red Action PrivilegeEscalate User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 17

Red Action PrivilegeEscalate User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 18

Red Action ExploitRemoteService User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 19

Red Action ExploitRemoteService User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 20

Red Action PrivilegeEscalate User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -0.1

**Turn 21**

Red Action DiscoverRemoteSystems OpSubnet

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -0.1

**Turn 22**

Red Action DiscoverNetworkServices Op_Server0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | Scan | No |

Reward -0.1

**Turn 23**

Red Action DiscoverNetworkServices Op_Host0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | Scan | No |
| Op_Server0 | None | No |

Reward -0.1

**Turn 24**

Red Action PrivilegeEscalate User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Analyse User2 | 0 |
| Op Agent | Block UserSubnet | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | Privileged shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -1.1

**Turn 25**

Red Action ExploitRemoteService Op_Host0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Restore User2 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -0.1

**Turn 26**

Red Action ExploitRemoteService Op_Server0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 27**

Red Action ExploitRemoteService User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 28**

Red Action ExploitRemoteService User2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 29**

Red Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 30**

Red Action PrivilegeEscalate User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

Total Reward: -1.6

## B.2 Game play for policy analysis in Phase 3 - Scenario 1: Green Agent

### Turn 0

| | | | |
|---|---|---|---|
| Red Action | None | | |
| Green Action | None | | |
| Defender Actions | None | | |
| State | Hostname | Activity | Compromised |
| | User1 | None | No |
| | User2 | None | No |
| | Op_Host0 | None | No |
| | Op_Server0 | None | No |
| Reward | 0.0 | | |

### Turn 1

| | | | |
|---|---|---|---|
| Red Action | DiscoverRemoteSystems UserSubnet | | |
| Green Action | Sleep | | |
| Defender Actions | Agent | Actions | Comms |
| | User Agent | Sleep | 0 |
| | Op Agent | Sleep | 0 |
| State | Hostname | Activity | Compromised |
| | User1 | None | No |
| | User2 | None | No |
| | Op_Host0 | None | No |
| | Op_Server0 | None | No |
| Reward | 0.0 | | |

### Turn 2

| | | | |
|---|---|---|---|
| Red Action | DiscoverNetworkServices User2 | | |
| Green Action | Sleep | | |
| Defender Actions | Agent | Actions | Comms |
| | User Agent | Sleep | 0 |
| | Op Agent | Sleep | 0 |
| State | Hostname | Activity | Compromised |
| | User1 | None | No |
| | User2 | Scan | No |
| | Op_Host0 | None | No |
| | Op_Server0 | None | No |
| Reward | 0.0 | | |

**Turn 3**

| | |
|---|---|
| Red Action | DiscoverNetworkServices User1 |
| Green Action | GreenPortScan Op_Host0 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Scan | No |
| User2 | None | No |
| Op_Host0 | Scan | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 4**

| | |
|---|---|
| Red Action | ExploitRemoteService User1 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 5**

| | |
|---|---|
| Red Action | ExploitRemoteService User2 |
| Green Action | GreenProcess User2 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Exploit | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 6**

Red Action Sleep

Green Action GreenPortScan User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Scan | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 7**

Red Action PrivilegeEscalate User2

Green Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Analyse Op_Server | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -0.6

**Turn 8**

Red Action DiscoverRemoteSystems OpSubnet

Green Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -0.1

### Turn 9

| | |
|---|---|
| Red Action | DiscoverNetworkServices Op_Host0 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | Scan | No |
| Op_Server0 | None | No |

| | |
|---|---|
| Reward | -0.1 |

### Turn 10

| | |
|---|---|
| Red Action | DiscoverNetworkServices Op_Server0 |
| Green Action | GreenPortScan User1 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Scan | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | Scan | No |

| | |
|---|---|
| Reward | -0.1 |

### Turn 11

| | |
|---|---|
| Red Action | PrivilegeEscalate User1 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Analyse User2 | 0 |
| Op Agent | Block UserSubnet | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | Privileged shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

| | |
|---|---|
| Reward | -1.1 |

**Turn 12**

Red Action ExploitRemoteService Op_Host0

Green Action GreenPortScan Op_Server0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Restore User2 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | Scan | No |

Reward -1.1

**Turn 13**

Red Action ExploitRemoteService User1

Green Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 14**

Red Action ExploitRemoteService User2

Green Action GreenProcess Op_Host0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Analyse User2 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | User shell |
| User2 | None | No |
| Op_Host0 | Exploit | No |
| Op_Server0 | None | No |

Reward -1.5

**Turn 15**

| | |
|---|---|
| Red Action | ExploitRemoteService User2 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 16**

| | |
|---|---|
| Red Action | PrivilegeEscalate User1 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Analyse User2 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | User shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -1.5

**Turn 17**

| | |
|---|---|
| Red Action | Sleep |
| Green Action | GreenPortScan User1 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User2 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Scan | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -1.0

**Turn 18**

Red Action: PrivilegeEscalate User2

Green Action: GreenPortScan Op_Server0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | Scan | No |

Reward: 0.0

**Turn 19**

Red Action: ExploitRemoteService User2

Green Action: Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Exploit | User shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward: 0.0

**Turn 20**

Red Action: ExploitRemoteService User1

Green Action: Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User2 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward: 0.0

**Turn 21**

| | |
|---|---|
| Red Action | Sleep |
| Green Action | GreenProcess User1 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

| | |
|---|---|
| Reward | -1.0 |

**Turn 22**

| | |
|---|---|
| Red Action | PrivilegeEscalate User1 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

| | |
|---|---|
| Reward | 0.0 |

**Turn 23**

| | |
|---|---|
| Red Action | PrivilegeEscalate User2 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

| | |
|---|---|
| Reward | 0.0 |

### Turn 24

| | |
|---|---|
| Red Action | ExploitRemoteService User1 |
| Green Action | GreenProcess Op_Host0 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | Exploit | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 25

| | |
|---|---|
| Red Action | ExploitRemoteService User2 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User1 | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | Exploit | User shell |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

### Turn 26

| | |
|---|---|
| Red Action | Sleep |
| Green Action | GreenProcess User1 |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Remove User2 | 0 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -1.0

**Turn 27**

Red Action PrivilegeEscalate User2

Green Action GreenProcess User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 1 |
| Op Agent | Block UserSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -1.0

**Turn 28**

Red Action PrivilegeEscalate User1

Green Action Sleep

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Analyse Op_Server0 | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 29**

Red Action ExploitRemoteService User2

Green Action GreenPortScan Op_Host0

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| Op_Host0 | Scan | No |
| Op_Server0 | None | No |

Reward 0.0

**Turn 30**

| | |
|---|---|
| Red Action | ExploitRemoteService User1 |
| Green Action | Sleep |

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Op Agent | Sleep | 1 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| Op_Host0 | None | No |
| Op_Server0 | None | No |

Reward -0.5

Total Reward: -10.8

## B.3 Partial Game play for policy analysis in Phase 3 Scenario 3: Large Network

### Turn 12

Red Action PrivilegeEscalate User3

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Enterprise Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| User3 | None | No |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | None | No |
| Enterprise4 | None | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -0.2

### Turn 13

Red Action DiscoverRemoteSystems EnterpriseSubnet

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Enterprise Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| User3 | None | No |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | None | No |
| Enterprise4 | None | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -0.2

**Turn 14**

Red Action DiscoverNetworkServices Enterprise3

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Enterprise Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| User3 | None | No |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | Scan | No |
| Enterprise4 | None | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -0.2

**Turn 15**

Red Action DiscoverNetworkServices Enterprise4

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Enterprise Agent | Sleep | 1 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| User3 | None | No |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | None | No |
| Enterprise4 | Scan | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -0.2

**Turn 16**

Red Action DiscoverNetworkServices Enterprise2

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Analyse User3 | 0 |
| Enterprise Agent | Block UserSubnet | 1 |
| Op Agent | Analyse Op_Server | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| User3 | None | Privileged shell |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | None | No |
| Enterprise4 | None | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -1.7

**Turn 17**

Red Action DiscoverNetworkServices Enterprise1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Restore User3 | 1 |
| Enterprise Agent | Block UserSubnet | 1 |
| Op Agent | Block EnterpriseSubnet | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | None | No |
| User2 | None | No |
| User3 | None | No |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | None | No |
| Enterprise4 | None | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -2.2

**Turn 18**

Red Action ExploitRemoteService User1

Defender Actions

| Agent | Actions | Comms |
|---|---|---|
| User Agent | Sleep | 0 |
| Enterprise Agent | Sleep | 0 |
| Op Agent | Sleep | 0 |

State

| Hostname | Activity | Compromised |
|---|---|---|
| User1 | Exploit | User shell |
| User2 | None | No |
| User3 | None | No |
| User4 | None | No |
| Enterprise1 | None | No |
| Enterprise2 | None | No |
| Enterprise3 | None | No |
| Enterprise4 | None | No |
| Op_Host0 | None | No |
| Op_Host1 | None | No |
| Op_Host2 | None | No |
| Op_Server0 | None | No |

Reward -0.1