IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 6 May 2024, accepted 23 May 2024, date of publication 27 May 2024, date of current version 7 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3406148

RESEARCH ARTICLE

A Reinforcement Learning Approach to Military Simulations in Command: Modern Operations

ADONISZ DIMITRIU, TAMÁS V. MICHALETZKY, VIKTOR REMELI, AND VIKTOR R. TIHANYI
Széchenyi István University, 9026 Győr, Hungary

Corresponding author: Adonisz Dimitriu (dimitriu.adonisz@techtra.hu)

This work was supported by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, through the "Nemzeti Laboratóriumok pályázati program" Funding Scheme under Project 2022-2.1.1-NL-2022-00012.

ABSTRACT This paper presents a Reinforcement Learning (RL) framework for Command: Modern Operations (CMO), an advanced Real Time Strategy (RTS) game that simulates military operations. CMO challenges players to navigate tactical, operational, and strategic decision-making, involving the management of multiple units, effective resource allocation, and concurrent action assignment. The primary objective of this research is automating and enhancing military decision-making, utilizing the capabilities of RL. To achieve this goal, a parameterized Proximal Policy Optimization (PPO) agent with a unique architecture has been developed, specifically designed to address the unique challenges presented by CMO. By adapting and extending methodologies from achievements in the domain, such as AlphaStar and OpenAI Five, the agent showcases the potential of RL in military simulations. Our model can handle a wide range of scenarios presented in CMO, marking a significant step towards the integration of Artificial Intelligence (AI) with military studies and practices. This research establishes the groundwork for future explorations in applying AI to defense and strategic analysis.

INDEX TERMS Reinforcement learning, military simulations, tactical and strategic AI, military decision making, command modern operations.

I. INTRODUCTION

Reinforcement learning has been a key area in machine learning (ML) demonstrating its ability to master a variety of tasks, learning through interaction with the environment. The initial triumphs in the domain were marked by the application of RL to Atari games [1], offering a valuable ground for the development of state-of-the-art algorithms. These initial environments paved the way for RL's progression into more intricate and challenging domains, such as the multi-agent, partially observable world of StarCraft II, now a key benchmark in the field [2]. In this evolving landscape, the Deep RTS simulator inspired by StarCraft II was introduced as a resource-friendly alternative for RTS simulations specifically designed for deep RL research [3].

The impressive achievements of RL agents, such as AlphaStar in StarCraft II [4] and OpenAI Five agent in Dota 2 [5], have demonstrated their capability to handle environments with huge state and action spaces, achieving super-human performance. Despite these milestones, there is a continuous need to expand the horizons of RL in the RTS gaming realm as each new RTS game presents unique challenges and offers valuable insights into ML algorithms.

A significant challenge in this domain is the computational resources required for training RL agents in large-scale RTS games. Creating agents with capabilities comparable to AlphaStar in StarCraft II necessitates extensive computational resources, often beyond the reach of many research setups. This high computational demand restricts the accessibility of advanced RL research to only those with substantial resources and becomes a limiting factor in the iterative process of experimentation and model development. In response, environments like MicroGym [6] have been introduced besides Deep RTS, targeting low-resource research. MicroGym offers a simplified setting where agents coordinate multiple workers to collect resources

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

and combat enemy units, encapsulating fundamental aspects of RTS games.

There has been an increased need for AI in military applications. While RTS games share many similarities with military simulations, including partial observability and multi-agent environments, they cannot fully replicate the specific demands of real-world military operations.

The current study addresses this gap by introducing an RL agent specifically designed for CMO, a complex RTS game in the wargaming genre. This agent is equipped to handle any scenario created within CMO. Utilizing CMO's scenario editor, the users can craft a variety of training environments. This feature is particularly beneficial as it enables the customization of scenarios to align with our computational resources.

CMO differs from traditional RTS games by focusing on tactical, operational scale, and strategic-level operations, rather than micromanagement. It presents a multi-agent, cooperative, partially observable environment, where multiple actions can be selected concurrently. While this aspect adds to the game's complexity, it also makes CMO a candidate platform for developing advanced RL agents, providing the opportunity to test different algorithms in a complex dynamic setting.

Our contribution to the field is twofold. Firstly, we have successfully developed a direct Application Programming Interface (API) for CMO. To the best of our knowledge, this is the first work to achieve a fully functional RL API for the game. This API facilitates the interaction of RL agents with CMO's simulation environment, expanding the horizon of RL in complex and realistic scenarios mirroring real-world military operations. Secondly, we have designed a novel neural network architecture inspired by notable RL achievements such as AlphaStar and OpenAI Five. This architecture is specifically designed to handle the multi-unit control and simultaneous multi-action selection complexities inherent in CMO.

II. RELATED WORK

The research of RL in military operations is considerably less extensive compared to other domains. A recent study [7] explores the potential of games and simulators for developing AI in military Command and Control (C2) tasks. They discuss the parallels between complex multi-agent scenarios in games and military operations, emphasizing the role of AI in military missions and impacting future battlefields. Reference [8] combines a Multi-Agent System simulator with a network emulator to create an accurate representation of battlefield environments.

Research in this area has been characterized by simplifications, such as the use of grid maps or limited and controlled environments. Reference [8] addressed the complexity and dynamic nature of modern military environments, by integrating multi-agent system simulators with network emulators. Their work emphasizes the critical role of communication networks in creating realistic military simulations.

One study simulated small-scale military engagements in a grid world, training policy gradient RL agents to combat static enemies on a 10×10 grid map [9]. Another research compared Deep Q-Learning (DQL) and Actor-Critic networks in simple ground combat scenarios, such as rendezvous and obstacle avoidance, performed on an 80×80 pixel map [10].

Further efforts have been made to improve multi-agent dynamics in military simulations. One notable work utilized an enhanced Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm, augmented with supervised learning, to train agents in grid-like environments with map features. This resulted in improved win rates in general attack scenarios compared to other algorithms [11]. Additionally, a PPO [12] agent was trained to perform air defense strategies in a digital battlefield created using Unreal Engine, surpassing traditional C2 systems in their specified tasks [13].

AI in military training has also attracted attention [14]. One study employed a graph neural network to develop adaptive non-player characters (NPCs) for military training simulations focusing on NPC cooperations through behavior prediction [15]. Another research integrated military doctrine with multi-agent RL (MARL) to enhance decision-making in simulations [16]. They demonstrated that the combined model outperformed both the doctrine-only and MARL-only methods using the StarCraft Multi-Agent Challenge (SMAC) environment [17].

Despite these advancements, effectively addressing the complexity of real-world military operations continues to be a significant challenge.

According to our knowledge, only a few publications have utilized CMO as an environment for their research. The authors in [18] conducted their research using CMO, specifically focusing on the simulation of various combat operations. Notably, they did not employ RL methods in their study. Instead, they utilized CMO to simulate 10,000 combat operations related to the warning and reconnaissance mission of an underwater manned/unmanned cooperative attack and defense system.

Another study involved pre-training an RL agent on a surrogate model before applying it in CMO for the straightforward task of maneuvering a unit to a specific location and evading enemy radar detection [19]. This research, while providing proof of concept, faced challenges in directly integrating with CMO due to the absence of a suitable API. Consequently, their approach necessitated a surrogate simulation environment model for training which is a strong limitation.

In contrast, this research has successfully developed an API that directly interfaces with CMO, enabling us to train an RL agent on a broader range of military scenarios. This advancement marks a significant step forward in the application of RL within complex wargaming environments.

Given the limited work in strategic military RL and the complexity inherent in CMO, researchers need to draw inspiration from sophisticated RTS game agents like

AlphaStar and OpenAI Five. For example [20] explored the application of a similar network to AlphaStar. Their work involved training the RL agent to function as a military commander in a combat simulation 'ReLeGSim', showcasing the agent's ability to make tactical decisions and manage units.

Both AlphaStar and OpenAI Five share several core similarities, reflecting the nature of the RTS games they were designed for. Both agents process in-game scalar and pixel-based observations using LSTM cores within their network. Their training involves asynchronous distributed RL frameworks, utilizing thousands of CPU cores and GPUs, to manage the extensive data and interactions of their game environments. A significant part of their training involves self-play, where the agents improve by competing against past versions of themselves.

Despite these similarities, AlphaStar and OpenAI Five have notable differences. AlphaStar's training includes a range of techniques such as TD(λ), V-trace [21], UPGO, an initial phase of supervised learning, and a MARL algorithm called the league. In contrast, OpenAI Five predominantly utilizes the PPO algorithm with Generalized Advantage Estimation (GAE) [22]. Architecturally, AlphaStar is more complex, integrating transformers, pointer networks, and Multi-Layer Perceptrons (MLPs), whereas OpenAI Five relies more on MLPs, increased computational resources, and extended training duration (10 months compared to AlphaStar's 44 days).

This work analyzed and integrated various elements from these advanced AI systems to make the most efficient use of available resources.

III. COMMAND: MODERN OPERATIONS GAME

A. INTRODUCTION TO COMMAND: MODERN OPERATIONS

CMO offers a comprehensive and detailed simulation of post-World War II to contemporary military operations across air, naval, and ground domains. The game provides an intricate platform for the application of real-world military strategies and tactics, underpinned by an extensive database of historical and modern military hardware and systems. The simulation engine is capable of handling a wide range of military engagements, from localized encounters to large-scale, global conflicts.

The game's graphical user interface features a comprehensive global view of the Earth, rendered with high-resolution satellite imagery and detailed terrain maps, providing the foundational environment for all in-game operations as shown in Fig. 1. Players have operational control of various military units, including aircraft, ships, submarines, ground forces, and even strategic weapons, navigating them through complex missions and scenarios.

Every unit within the game is modeled with high fidelity to real-world specifications, covering aspects such as weapon capabilities, fuel consumption, physical limitations, sensor functionality, and authentic communication systems, ensuring a highly accurate simulation.

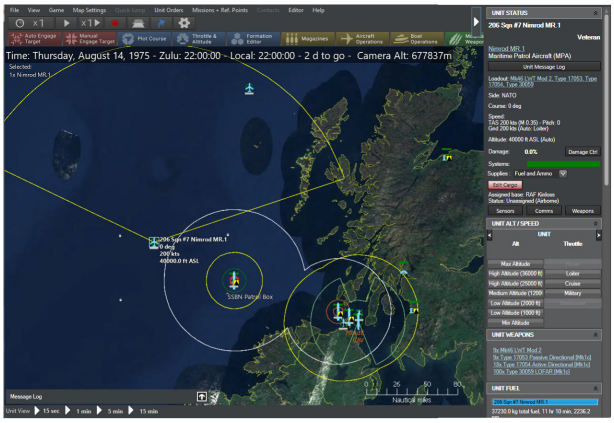


FIGURE 1. Command: Modern operations.

CMO is equipped with a scenario editor that allows players to create different scenarios, from historical battles to virtual conflicts, providing the means for examining complex military operations. This makes CMO not just a platform for entertainment, but also a tool for military training and strategic analysis [18].

The integration of CMO with an RL agent builds upon the initial work found in [23]. Despite this starting point, considerable challenges were encountered in advancing the project. The task of efficiently setting up the API (a crucial step for ensuring fast execution and effective training), demanded substantial effort. The multifaceted nature of the game provides an ideal testbed for advanced AI agents, making our efforts highly beneficial.

B. GAME MECHANICS AND DYNAMICS

This RTS game features multi-sided scenarios, where the player has the control of managing any side within a conflict. Each side comprises several units and resources that must be strategically managed to achieve scenario-specific objectives.

While multiplayer mode is available, CMO is primarily played in a single-player format - often involving the management of multiple sides in adversarial and cooperative scenarios - similar to a self-played chess game. When controlling a single side, the game presents a partially observable environment, where information is limited to the perspective of that side. While the player can switch views and gain insights into different sides, each side independently maintains its own limited and context-specific field of observation. This property necessitates the use of recurrent networks like LSTM.

The game operates in real-time mode, offering the flexibility to pause and resume the simulation at any moment, with the added option for accelerated play. This feature enables rapid execution, a key factor for efficiently training an RL agent. In practice, the agent will interact with the game in a style similar to turn-based strategy games.

Specifically, upon pausing the game, the player or the RL agent can simultaneously select multiple actions for each unit of a side. This includes selecting actions, such as moving, attacking, refueling, etc. (see in Table 6). This unique characteristic distinguishes CMO from other games and necessitates a slightly different architecture. This architecture must account for the simultaneous prediction of multiple actions per unit, representing a significant departure from the more linear action selection processes observed in other environments [4], [5].

The game is periodically paused at set intervals, such as every 5 minutes to allow the agent to evaluate the situation and make decisions. Importantly, each scenario has a defined time span, and the game concludes either when this duration elapses or when the primary objectives of the scenario are met. A typical scenario could involve anywhere from 10 to 1000 turns or interactions, though the API has no strict upper limit.

C. GAME COMPLEXITY AND CHALLENGES

CMO models realistic damage mechanics and resource management, including ammunition, fuel, and unit repair. It provides a diverse range of geographical landscapes, from dense urban areas to seas and rugged terrain. The game simulates real-world environmental conditions, such as weather patterns, day-night cycles, and seasonal changes, which impact visibility, unit mobility, and sensor effectiveness. For instance, weather can affect sensor performance and unit movement, while terrain impacts visibility and mobility. CMO features advanced sensor and detection systems ranging from simple reconnaissance equipment to sophisticated radar and sonar systems that can be affected by these environmental factors.

CMO has a large database, which includes thousands of intensely detailed sensors, weapons, ground units, aircraft, ships, submarines, and facilities. This diversity could raise a significant challenge for the agent, requiring it to generalize and adapt to the specific capabilities and limitations of many equipment and technologies.

In CMO, while players have the flexibility to pause and resume gameplay at any time, a decision was made to implement a turn-based approach for agent interaction. This method involves pausing and resuming the game at predetermined fixed intervals, which are manually set for each scenario. This allows the agent to process and respond to in-game events in a structured manner while acknowledging that alternative methods could also be viable. The length of these intervals (further referred to as the "simulation step") depends on the temporal scope of the scenario. For instance, in scenarios with a condensed timeline, requiring rapid and frequent decision-making, the interaction interval might be 5 seconds. On the other hand, in more extended scenarios, where strategic developments unfold over a longer period, an interval of five minutes may be sufficient to enable meaningful agent intervention without overlooking critical events within the game.

Regarding the built-in AI functionality, CMO provides an option for AI to control a side, but documentation on its capabilities is sparse. Through experimentation, it was observed that the built-in AI strategy is limited to executing auto-attacks upon detecting enemy units. Given this constraint, this research neglects the use of the built-in AI and relies purely on self-play.

In this study, the decision was made to not utilize the built-in AI functionalities of CMO. This choice was based on the alignment of the AI's characteristics with the specific requirements of our experimental setup. Instead, the focus was on developing and utilizing a self-play framework to fully control the training and evaluation.

IV. PRELIMINARIES

PPO is a widely adopted policy gradient method that has achieved success in a variety of tasks. It is a successor of Trust-Region policy gradient (TRPO) [24] and is considered one of the state-of-the-art algorithms. The choice of the PPO algorithm, an on-policy learning method, was guided by several considerations. Firstly, PPO offers a balance between simplicity of implementation and stability of training, which is particularly important given the complex nature of the game. The decision was also influenced by the empirical success of PPO in similar domains, including other RTS games and multi-agent environments. The algorithm is also proven to be effective in managing large state and action spaces.

The algorithm's key innovation is the clipped surrogate objective function, which essentially prevents the policy from moving too far from its current state.

For a given a policy π_θ parameterized by θ and an action a_t in state s_t, the policy's probability ratio $r_t(\theta)$ is defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{1}$$

that expresses the change in policy probability due to updating parameters. Using this ratio, the PPO objective can be formulated as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2}$$

where ϵ is the clipping threshold. The advantage estimation $\hat{A}_t$, is calculated using GAE, which aims at balancing the bias-variance trade-off in advantage computation, smoothing the policy updates. The GAE is computed as follows:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{3}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ represents the temporal difference error, γ is the discount factor, λ is the GAE parameter, and T denotes the simulation horizon. To further improve the training stability and reduce the risk of large policy deviations, a Kullback-Leibler (KL) divergence penalty is integrated into the objective function:

$$L^{\text{KL}}(\theta) = \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot|s_t) \parallel \pi_\theta(\cdot|s_t) \right] \right], \tag{4}$$

This penalty is to further ensure, that the updated policy remains in close proximity to the old policy, preventing drastic changes that could harm the training process.

The final objective function combines these elements along with a value loss term and an entropy bonus $\mathbb{H}$ to encourage exploration:

$$L(\theta) = \hat{\mathbb{E}}_t[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) - c_2 L^{\text{KL}}(\theta) + c_3 \mathbb{H}[\pi_\theta](s_t)], \tag{5}$$

where $L^{\text{VF}}(\theta)$ is the squared-error loss $(V_\theta(s_t) - V_t^{targ})^2$ and c_1, c_2, c_3 are the coefficients for each term in the loss function.

V. ARCHITECTURE

This section presents the neural network architecture of the PPO agent, designed to play any scenario in CMO. A simplified representation of the RL agent is illustrated in Fig. 2, showcasing a shared network that processes the observation through embeddings. This design is significantly influenced by the architectures of OpenAI Five and AlphaStar. It should be noted that replicating or adapting their model is far from straightforward, given the complexity and depth of their original architectures.

The input is structured into three categories. The first category consists of scalar inputs, which include scenario-specific information such as current time, number of units lost, and number of contacts defeated. Unlike AlphaStar and OpenAI Five, our model does not incorporate pixel-based observations. Instead, entities in the scenario are categorized into two groups: 'units' (own units) and 'contacts' (enemy units), as they are commonly referred to in the game and illustrated in Fig. 3

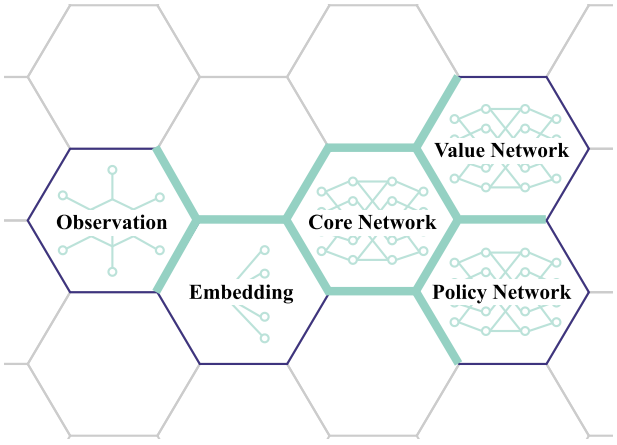


FIGURE 2. Simplified network architecture.

Further drawing from AlphaStar, transformer models are employed to encode entity-type observations. These entity encodings are then aggregated through a max pooling operation and subsequently concatenated with the output of the scalar encoder. This combined data is fed into an LSTM network, as depicted in Fig. 3. The network's value function is determined by a simple MLP that processes the LSTM output.

The architecture of the action head is more complex, comprising two primary components: the action type head and the action argument head. This design of the action-selection module tries to balance complexity with functionality, as shown in Fig. 4 and described in section V-B in more detail.

One of the key modifications in the architecture is its capability to manage the multi-unit control dynamics of CMO. Unlike traditional RTS games where the focus might be on singular unit control or smaller groups, CMO requires the simultaneous coordination of multiple units of a side elevating the problem to a MARL challenge. The network is designed in a way to handle the variable number of units with transformers. The transformers allow the network to dynamically adjust its focus and resource allocation based on the situational demands of each unit in the environment.

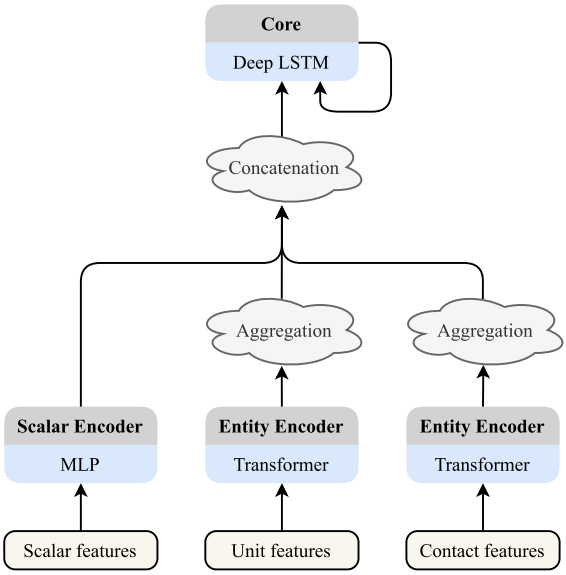


FIGURE 3. Core network.

Another aspect the PPO agent addresses is the multi-action selection feature inherent in CMO. In the context of CMO, when gameplay is paused, a player can assign a set of actions to each unit; for example, navigating to a designated location, adjusting the unit's speed, firing a weapon to a particular target, activating radars, deactivating sonars. These are executed simultaneously once the simulation is resumed. This multi-action selection mechanism differs from the traditional RTS environments, where actions are typically sequential or limited in parallel execution [4], [5]. Our agent is engineered to enable units to execute multiple actions concurrently. To accommodate this, the agent is engineered to output multiple actions for each unit concurrently.

This architecture not only addresses the current requirements of CMO gameplay but also provides a scalable framework capable of adapting to more complex scenarios and future expansions.

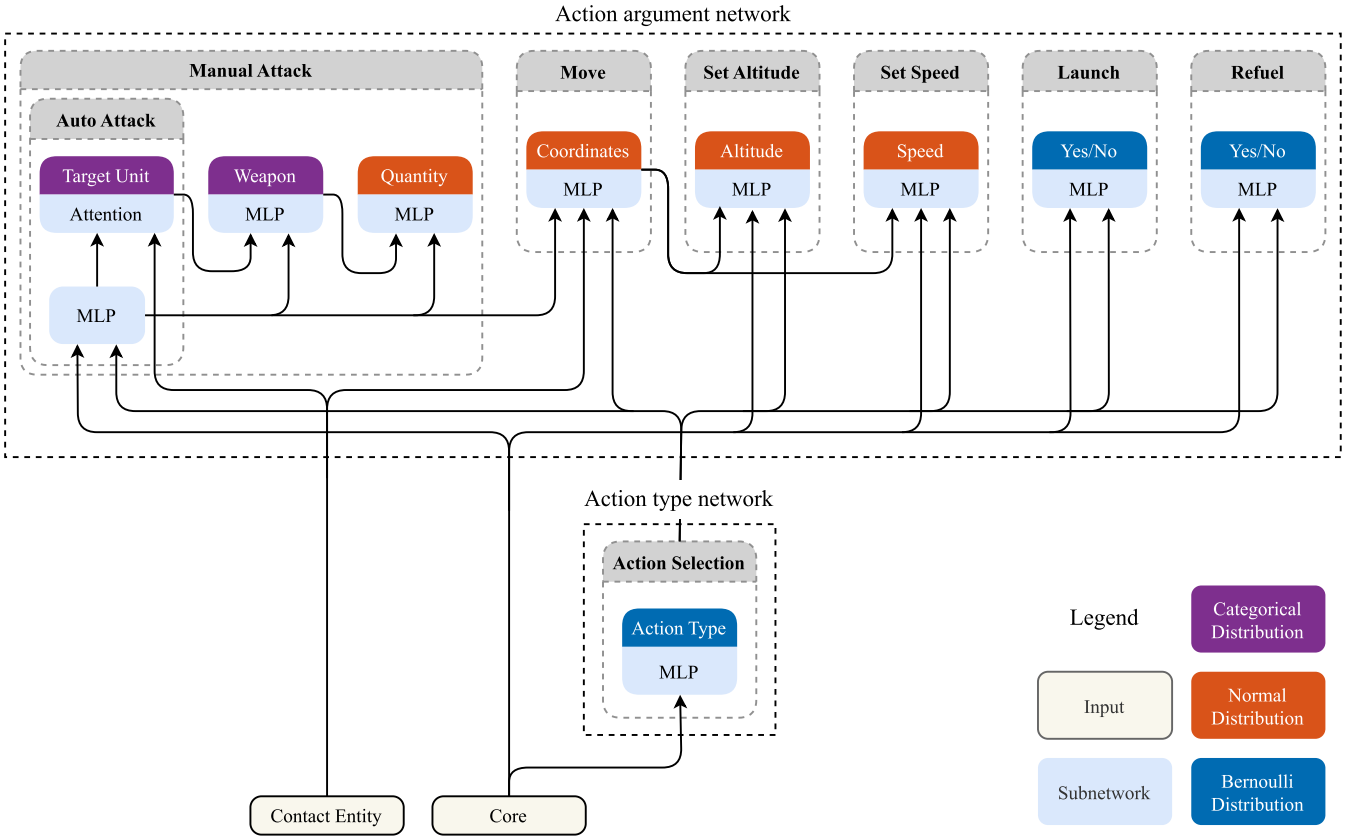


FIGURE 4. Policy network.

A. OBSERVATION SPACE REPRESENTATION

In the development of the PPO agent's observation space, a structured observation data approach was adopted. In the context of CMO, the choice of using structured data over pixel data is quite strategic and is informed by the following key considerations:

- CMO's gameplay interface offers a large amount of data, accessible through interactive elements such as unit selections or menu buttons. These interactions trigger a display window that provides additional information about the selected units, resources, or game elements. This method provides access to a depth of information that is not readily available in the GUI's visual representation. Therefore, relying solely on pixel information would be insufficient, as they do not capture the full spectrum of data necessary for the agent's decision-making processes.
- Traditional RL agents that play video games are usually trained using pixel data [1], [2], [5]. However, CMO can operate without graphical rendering through Command Line Interface which presents a substantial advantage for employing structured data. This not only reduces the computational load but also reallocates more GPU capacity for tensor operations, which is particularly beneficial for setups with limited resources.
- CMO currently does not natively support the extraction of pixel data from the game or GUI. This limitation makes it impractical to use pixel data as the primary source of information for RL agents.

Unlike pixel-based observations which is a visual snapshot of the game state, structural observations require explicit, programmatic extraction and processing of game data. This procedure requires careful coding and integration to ensure accurate and efficient extraction of the relevant information from the game's interface.

The observation space for each unit in the game is composed of a set of features, such as unit health, position, heading, weapon information, etc. A complete list of features is shown in Table 3 for units, and in Table 5 for contacts.

B. ACTION SPACE DESIGN

Actions can be executed through the game's Lua API, which, despite having a limited set of commands, is versatile enough to replicate nearly all functionalities accessible through the game's GUI. 13 fundamental actions commonly used in a typical game scenario have been identified and implemented, as listed in Table 6. Each action requires a unique combination of arguments, which vary in type - from categorical, discrete, and continuous - and in the number of arguments needed. This kind of complexity is typical of

advanced games. This is addressed in the model by dividing the action space into two primary prediction tasks: action type prediction and action argument prediction.

Generally, there are two basic approaches to implementing this action composition. The first involves predicting arguments in a single vector or "head", assigning a specific position in the vector for each argument. The other one, which we have adopted, involves using distinct 'heads' for each action argument. This means the action arguments of each action type are predicted by a separate, specialized component in the model.

Both the action types and their corresponding arguments are treated as independent of each other. While this assumption may not fully reflect the intricate interdependencies in reality, it greatly simplifies the calculation of the policy gradient. Given the action a_t at time step t, it can be decomposed into several distinct action types, denoted as $a_t^1, a_t^2, \ldots, a_t^n$. Each action type a_t^i is further associated with its own set of arguments $a_t^{i,1}, a_t^{i,2}, \ldots, a_t^{i,m_i}$. The probability of taking a_t given state s_t, represented by $\log \pi_\theta(a_t|s_t)$, is computed by summing the log probabilities of each action type and its arguments as follows:

$$\log \pi_\theta(a_t|s_t) = \sum_{i=1}^{n} \left(\log \pi_\theta(a_t^i|s_t) + \sum_{j=1}^{m_i} \log \pi_\theta(a_t^{i,j}|s_t) \right) \tag{6}$$

C. ACTION SELECTION HEAD

The action type network is designed to output a binary value for each action type, utilizing a sigmoid activation function. The binary output essentially represents a decision on whether to execute a particular action or not. When an action is selected (indicated by the network outputting a '1'), the corresponding action's arguments are also chosen. As pointed out by many [2], [5], [6] action masks are important factors for efficient training in large action spaces. In the action-type network, action masks are carefully implemented and employed to dynamically adjust the network's output space based on the current state and feasibility of actions. For example, if no contact is available for attacking, then the attack entry in the action mask is set to zero, effectively disabling this option in the network's decision process and allowing the network to focus on other feasible actions.

D. ACTION ARGUMENTS HEADS

In contrast to simpler games where a single argument head might suffice for all the actions (for example which specifies the direction at which the action is issued) such as in [4], [5], and [20], CMO's parameterized actions as well as the multi-action selection nature requires a more tailored approach. In this work, distinct heads have been implemented for each action type, with each head responsible for predicting the specific arguments required for that action. While it requires manual configuration, it offers the flexibility to connect and integrate different modules within the action head and allows for a more nuanced and accurate prediction of action parameters as outlined in [25]. The selected action type is incorporated as input to each argument head. As noted in [26], when continuous actions depend on a discrete choice, the discrete action can serve as an input to the continuous one. However, this setup prevents gradients from flowing back through the sampled input. They propose a hybrid TRPO algorithm as a solution. In our case, since action types effectively act as masks for the arguments, there is no need for gradient flowback in this simple context. While this work have not yet explored alternative methodologies, this design choice lays the groundwork for future research. Subsequent studies may delve into different implementations, hoping to uncover more efficient or effective ways to handle the action space of CMO.

E. INTEGRATION WITH GAME ENVIRONMENT

The integration with the game environment operates through a classic RL loop depicted in Fig. 5. At fixed intervals, the game simulation is paused and the observation is extracted from the game's internal state. This observation is then processed, and a scenario-specific reward is calculated. The agent generates its actions based on the observation, which are fed back into the game environment.

It is important to highlight that certain features are not directly accessible through communicating with the game. Static features like unit type, the properties of sensors and weapons, are stored within CMO's internal database. Since these features remain constant during gameplay, fetching them from the database in each step would be impractical and would introduce undesired latency during rollouts. Therefore these features are obtained before training, through our DB interface as shown in Fig. 5

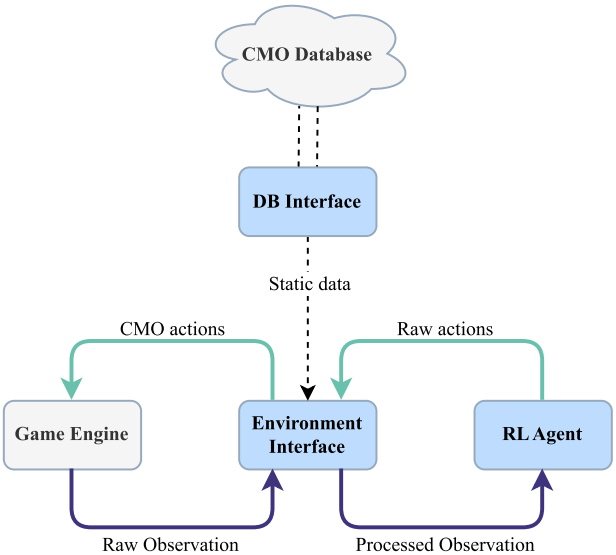


FIGURE 5. RL loop with a DB interface module.

Each scenario in CMO can present unique objectives, ranging from straightforward tasks like annihilating enemy

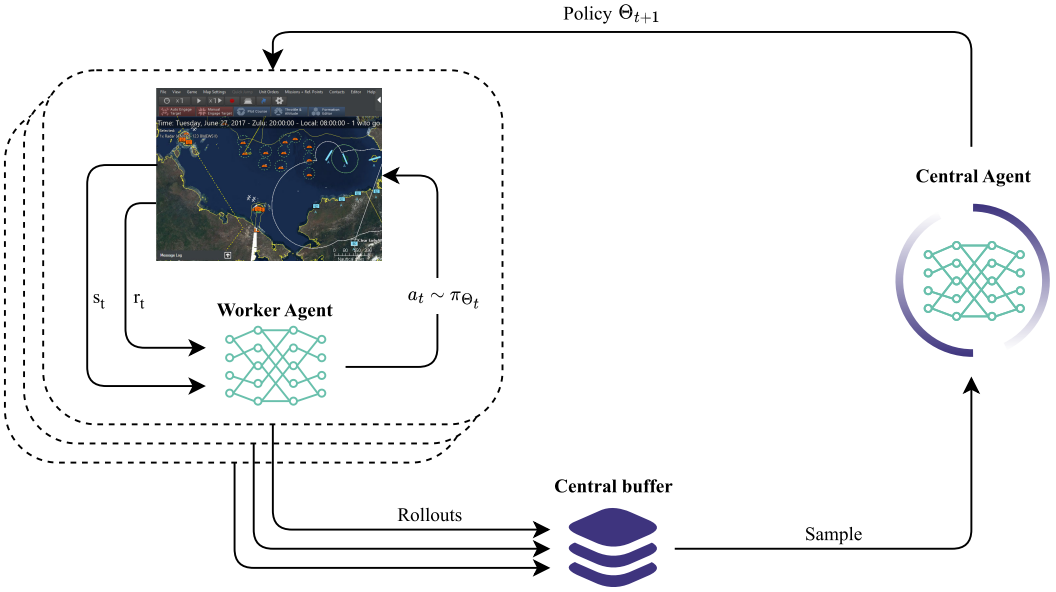


FIGURE 6. Distributed training with multiple workers.

units or defending an airbase to more intricate missions such as safely extracting a unit from a hostile environment or liberating a coalition side under military oppression. These diverse objectives pose a challenge in defining a universal reward function applicable across all scenarios. Consequently, for each new scenario introduced to the agent, a reward function must be manually defined to ensure that the reward signal aligns with the specific objectives of the scenario at hand.

VI. TRAINING AND IMPLEMENTATION

During the training and implementation phase of the PPO agent, various strategies have been explored to optimize the training process for speed and efficiency, enabling us to conduct the training on a single PC.

To enhance the model's performance the input features are normalized and the categorical variables are embedded, such as unit types and sub-types. For the LSTM networks backpropagation through time (BPTT) [27] is implemented, as a reasonable scenario is expected to be no more than 100-200 steps. For sufficient exploration, entropy regularization is incorporated in the loss function.

KL divergence was another component of the training methodology. It was utilized by incorporating it into the loss function to further regulate the policy updates as mentioned before, which resulted in a more stable learning trajectory.

Beyond its role in the loss function, KL divergence was also utilized as an early stopping criterion and as a learning rate scheduler:

$$\alpha = \begin{cases} \alpha_{\text{old}} \cdot \eta_\alpha, & \text{if } \mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t) \parallel \pi_\theta(\cdot|s_t)] > \mathrm{KL}_{\text{thres}} \\ \alpha_{\text{old}}, & \text{otherwise} \end{cases} \tag{7}$$

where α represents the learning rate, and η_α denotes the learning rate decay factor, with $0 < \eta_\alpha < 1$. The learning rate is adjusted only if the KL divergence between the new and the old policy exceeds a predefined threshold $\mathrm{KL}_{\text{thres}}$. This mechanism ensures that the learning rate is dynamically scaled down to prevent large policy updates that could destabilize the training.

This approach stemmed from the observation that KL divergence tended to increase as the agent's performance improved, due to higher rewards and consequently larger gradients. As the agent advanced in its training, the learning rate set at the beginning of training gradually became too aggressive for the later stages. By dynamically adjusting the learning rate in response to KL divergence levels, the training intensity keeps moderated, ensuring that the learning rate remains appropriate, and proportionate to the agent's development.

To significantly accelerate the training process of the PPO agent for CMO, a centralized distributed training framework is developed. In this setup, multiple agents known as 'worker agents' are tasked with collecting experiences in parallel. Meanwhile, a central agent accumulates these experiences, performs the parameter update, and sends the updated parameters back to the workers in a synchronized way. Instead of each worker performing a predetermined number of episodes - such as in [28] where actions are applied in batches across a fixed number of experiences -, a more dynamic workload distribution is used. Once a worker completes an episode, it decrements a shared episode counter, until the collective episode count for a parameter update is reached. The architecture of this framework is illustrated in Fig. 6.

Lastly, given the computational resource constraints commonly faced by researchers, the promotion of the curriculum

learning approach is suggested for scenarios with high complexity [29]. This method allows us to progressively train the agent on increasingly challenging tasks, making the learning process more effective despite limited resources.

VII. EXPERIMENTAL SETUP

To assess the performance of the model and determine appropriate hyperparameter intervals, a fundamental scenario is established for testing. This scenario features two opposing sides, each equipped with an Unmanned Aerial Vehicle (UAV) and an artillery unit. The primary objective is to locate the enemy's artillery using the UAV and then eliminate it using one's artillery. The task unfolds within a designated area, with each side's starting position randomized in every episode to test the agent's generalization capability across the entire area. The details of this scenario are outlined in Table 1.

TABLE 1. Scenario details.

Scenario feature	Value
Map Size	90 km x 90 km
Simulation step	5 min.
Scenario Duration	2 hours 55 min.
Max. episode length	35
Units	Skylark I-LEX UAV
	2S5 Giatsint-S SP Artillery

The UAV is equipped with a 180° field of view sensor with a range $r_{\text{range}} = 6.5$ km and can operate within a speed limit of 120 km/h to 166 km/h

A. SCENARIO ANALYSIS

The reason for this scenario is the deduction of a simple heuristic behavior that solves the task with nearly optimal effectiveness. The task can be naturally segmented into two phases: surveillance and elimination. Initially, the UAV is responsible for locating the enemy unit. Once the target is identified, the artillery must position itself within weapon range for successful elimination. This translates to an area exploration task for the UAV, which is a spiraling movement in a continuous world. However, due to the discrete simulation step (5 min. per interaction), the optimal pattern changes. Fig. 7 illustrates an approach, where an optimal hexagonal tiling of the map is assumed. Each hexagon's center is strategically positioned at a distance of $2r_{\text{range}}$ from its neighbors, aligning with the maximum range of the UAV's sensors.

The hexagon edges thus represent the outer boundary of the UAV's detection range, with the inner circle of each hexagon corresponding to the UAV's sensor range. The nearly optimal trajectory, which involves some overlapping, is depicted in Fig. 7. It involves moving between hexagon centers, and upon completing a circle, transiting to the closest center in the next hexagon layer. Such trajectory ensures a quick average detection time of an unknown enemy unit, given

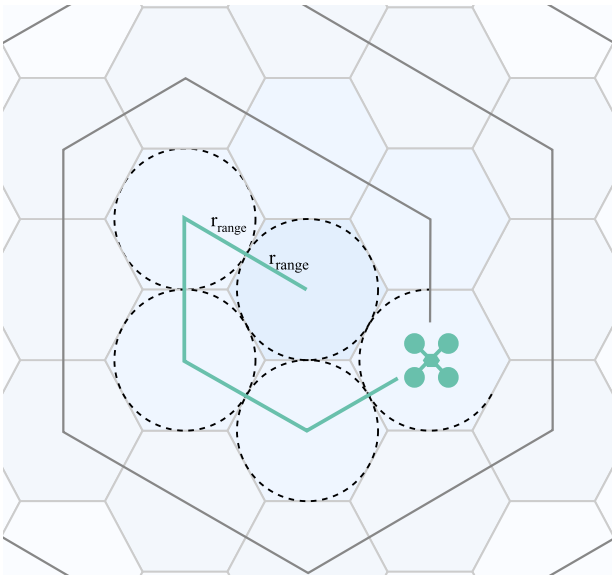


FIGURE 7. Hexagonal tiling-based exploration.

the constraints of the simulation step, the UAV's speed, and sensor range.

It is worth noting, that hexagonal tiling has been utilized in other research for area exploration tasks [30], where agents typically have six discrete movement options, corresponding to moving into one of the adjacent hexagonal cells. In this work, this hexagonal tiling serves as a theoretical baseline for analyzing the task. However, our approach differs as the agent is designed to predict continuous actions, including the coordinates to move. If the agent learns to mirror this movement pattern, it indicates a successful adaptation to the task. Additionally, the task of elimination proves to be straightforward. Due to the artillery's weaponry, the enemy can be eliminated within a single attack cycle.

B. REWARD FUNCTION

The reward function is structured to encourage behavior aligned with this strategy, utilizing the notation A_i to represent the area observed by the UAV at time step i. For the surveillance task, the reward at any time step t is given by:

$$r_t^{\text{surveillance}} = \frac{A_t^{\text{new}}}{A^{\max}} \cdot \exp\left(-\frac{d_t^{\text{base}}}{d^{\max}}\right) \cdot \exp\left(-\frac{t}{T}\right) \quad (8)$$

Here $A^{\max} = r_{\text{range}}^2 \pi$ represents the maximum observable area by the UAV at any given moment, used to normalize the reward. $A_t^{\text{new}} = A_t \setminus (\bigcup_{i=0}^{t-1} A_i)$ indicates the newly discovered area at time step t not overlapped with previously observed areas. d_t^{base} denotes the distance of the UAV from its starting position, $d^{\max}$ is the max length of the exploration area to normalize distances, and T is the maximum length of an episode. This formula rewards the discovery of new areas, incentivizing the agent to prefer closer points and explore them in the right sequence.

Upon successful detection or elimination of the enemy target, a fixed reward $r_t^{\text{detection}} = 1$ and $r_t^{\text{elimination}} = 2$, is awarded to reinforce these critical outcomes.

Once the enemy has been detected, the task changes from surveillance to elimination. The reward $r_t^{\text{closing}} = -d_{\text{target}}/d^{\max}$, penalizes the distance to the target, encouraging the artillery to get closer to it.

The total reward at time t, r_t is a composite of the individual rewards, shaping the agent's behavior towards effective surveillance and elimination.

$$r_t = r_t^{\text{surveillance}} + r_t^{\text{closing}} + r_t^{\text{detection}} + r_t^{\text{elimination}} \quad (9)$$

For the above scenario, 12 CPU cores and a GeForce RTX 4090 GPU of a single machine were utilized to conduct the training.

VIII. RESULTS

The outcomes of the training sessions, using the hyperparameter settings detailed in Table 7, are illustrated in Fig. 8. The agent underwent training utilizing shaped reward as defined by Formula (9), a sparse reward scheme (awarding +2 for successful enemy elimination), and through curriculum learning. Consistent with existing literature, our findings confirm that using the shaped reward results in more effective learning compared to the sparse reward setup. Specifically, the sparse reward scenario achieved an average success rate of 40% after 2 million environment steps (around 4 hours of training). Conversely, both the shaped reward and curriculum learning approaches reached an average success rate of around 90%. In the curriculum learning approach, the agent was initially trained on the surveillance component of the task using the reward specified in Formula (8). This pre-trained agent then was further trained on the full task using the reward in Formula (9) to adapt its policy to the elimination task as well. This allowed the agent to learn the elimination more easily and effectively as the learning curve suggests.

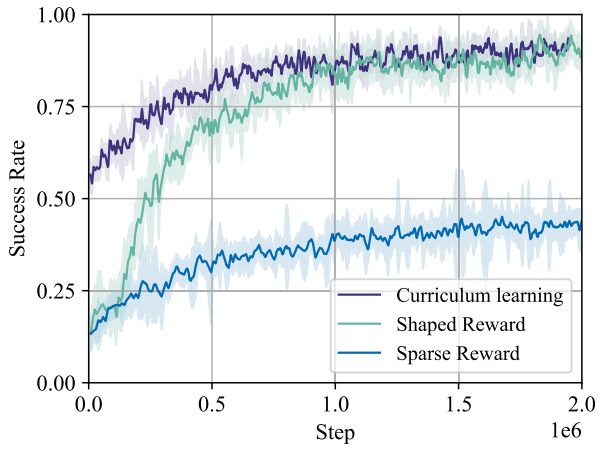
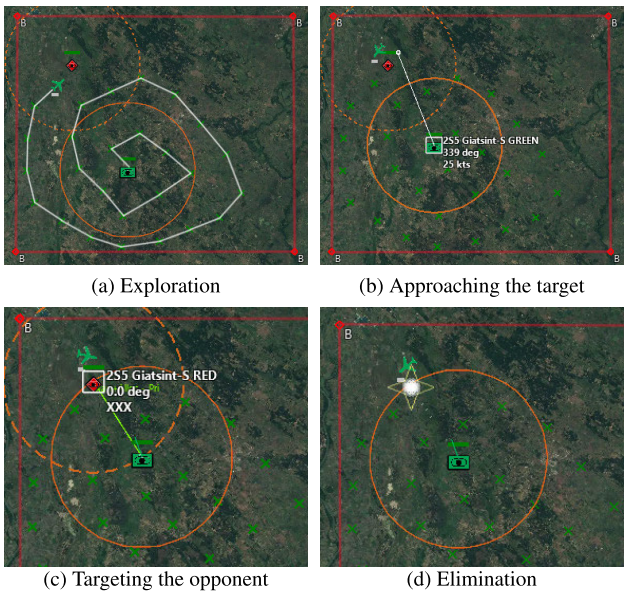


FIGURE 8. Progression of scenario success rates (averaged across 3 runs) under different training methodologies.

It was discovered that the learning rate and the loss coefficients were critical hyperparameters that significantly influenced performance. Adjustments to the entropy and KL divergence coefficients resulted in noticeable changes in the success rate. Additionally, the learning rate, the KL divergence coefficient, and the parameters of the proposed KL divergence-based learning rate decay (Formula 7) required intensive experimentation to achieve a balance between fast convergence and stability. A relatively large initial learning rate combined with a small KL divergence threshold led to a rapid decay in the learning rate and therefore convergence speed. On the other hand, when paired with a larger threshold, this configuration resulted in unstable learning outcomes. It has turned out that an appropriate KL threshold and decay parameter results in reasonably stable learning, even with a relatively high learning rate. This highlights the need for precise tuning in environments characterized by high-dimensional state and action spaces.



(a) Exploration (b) Approaching the target

(c) Targeting the opponent (d) Elimination

FIGURE 9. Main phases of completing the scenario.

Fig. 9 captures the completion of the task by the agent, which was trained on the shaped reward in a randomized setup. Specifically, Fig. 9a illustrates the agent's initial exploratory behavior. While the UAV does not strictly follow the hexagonal trajectory, it signifies the agent's capability to develop exploratory movements for detecting unknown enemy units. Following the identification of an enemy unit, the agent advances towards its target (Fig. 9b), engages in combat (Fig. 9c), and successfully neutralizes the opponent (Fig. 9d).

Interestingly, the agent trained through curriculum learning demonstrated an improved spiraling movement, as revealed in Fig. 10

These results emphasize the efficiency of curriculum learning even in this simple setup and highlight the adaptability of the agent in learning exploration and engagement strategies within CMO.

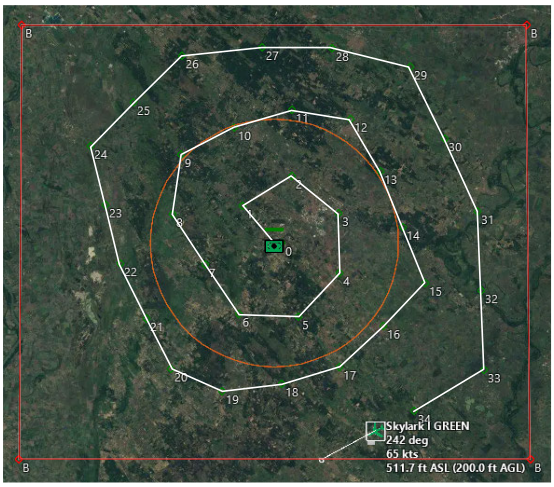


FIGURE 10. Trajectory learnt by curriculum learning.

IX. CONCLUSION

This paper described the RL framework for CMO, an RTS game for simulating military operations. Successfully developing an API for the game has enabled the agent to train directly from in-game observations. This API offers a novel tool for strategic military analysis and training. Our PPO agent demonstrated its ability to generalize in a surveillance and elimination task, where it has learned to perform the area exploration on a close optimal trajectory. This initial work provides an important platform for further research into AI-driven military strategy.

Looking ahead, there is potential in refining and expanding the agent's architecture, exploring more complex scenarios, and integrating advanced learning techniques to further improve performance. Moreover, future work will investigate the agent's strategic depth and temporal adaptability by allowing the agent to have precise control over action timing. This would enable the agent to pause and choose the timing of its actions with precision, mirroring the way a human player would interact with the game at their discretion.

APPENDIX
INSIGHTS INTO OBSERVATIONS AND TRAINING MECHANISMS

A. OBSERVATION

The observation space is designed to cover the critical attributes necessary for the agent's decision-making process. Scenario-specific attributes and general information are collected, which can be seen in Table 2. For each unit within the game, a comprehensive set of features is extracted, as detailed in Table 3. Each unit is capable of carrying up to four distinct weapons, each characterized by specific attributes such as effective ranges, outlined in Table 4. Additionally, essential data on enemy units (contacts) are gathered, detailed in Table 5. It is important to note that the information on contacts is restricted to observable characteristics, mirroring the limited information accessible to players during gameplay. This includes the exclusion of details such as weapon information.

TABLE 2. Scalar features extracted from the game.

Scalar feature	Description
ScenTime	Elapsed time in scenario
ScenDuration	Total scenario length
SimTime	Simulation step
NumDefeated	Enemy units defeated
NumDied	Friendly units lost

TABLE 3. Features extracted for each unit in CMO.

Unit feature	Description
UnitIdx	Unique identifier for each unit
SideIdx	Identifier for the unit's side
UnitType	Type of the unit
UnitSubType	Subtype of the unit
DamagePercent	Percentage of damage sustained
HeadingCos	Cosine of the unit's heading
HeadingSin	Sine of the unit's heading
CurrSpeed	Current speed
CurrAlt	Current altitude
Lon	Longitude position
Lat	Latitude position
CurrentFuelRatio	Current fuel ratio
MinAlt	Minimum altitude capability
MaxAlt	Maximum altitude capability
Weapons	Weapon information
RadarFlag	Radar system status
SonarFlag	Sonar system status
OECMFlag	Electronic countermeasure status
Condition	Operational condition of the unit
Base	Base of the unit
LastActions	Actions executed in the last step

TABLE 4. Features extracted for each weapon in CMO.

Weapon feature	Description
WeapType	Type of the weapon
LandRangeMin	Minimum range of land target
LandRangeMax	Maximum range of land target
AirRangeMin	Minimum range of air target
AirRangeMax	Maximum range of air target
TargetSpeedMin	Minimum speed of the target
TargetSpeedMax	Maximum speed of the target
TargetAltitudeMin	Minimum altitude of the target
TargetAltitudeMax	Maximum altitude of the target
RateOfFire	Weapon's rate of fire
QuantRemaining	Remaining ammunition in weapon
MaxQuant	Maximum ammunition
QuantInMag	Ammunition in magazine

B. TRAINING FRAMEWORK

The training approach relies on carefully chosen hyperparameters (see in Table 7) and techniques to enhance the

TABLE 5. Features extracted for each contact in CMO.

Contact feature	Description
ContactIdx	Unique identifier for each contact
SideIdx	Side of the contact
ContactType	Type of the contact
ContactSubType	Subtype of the contact
DamagePercent	Percentage of damage
HeadingCos	Cosine of the contact's heading
HeadingSin	Sine of the contact's heading
CurrSpeed	Current speed of the contact
CurrAlt	Current Alt of the contact
Lon	Longitude position of the contact
Lat	Latitude position of the contact
DetectAge	Elapsed time since last detection

TABLE 6. List of implemented actions.

Action Name	Argument size	Type
AutoAttackContact	1	categorical
ManualAttackContact	3	mixed
SetUnitCourse	2	continuous
SetAltitude	1	continuous
SetSpeed	1	continuous
Launch	1	boolean
Refuel	1	boolean
Rtb (Return to Base)	1	boolean
SetBase	1	categorical
SetRadar	1	boolean
SetSonar	1	boolean
SetOECM	1	boolean

TABLE 7. Hyperparameters of the training process.

Hyperparameter	Value
N_{workers} Number of workers	12
N_{ep} Total number of episodes	200
N_{batches} Number of mini-batches	2
N_{epochs} Number of update epochs	20
N_{steps} Number of steps collected	3500 - 7000
γ Discount Factor	0.99
λ for GAE	0.95
ϵ_{start} Initial clipping coefficient	0.2
ϵ_{end} Final clipping Coefficient	0.12
KL_{thres} KL threshold	0.05
α Learning rate	0.0002
η_α Learning rate decay factor	0.95
c_1 Value loss coefficient	0.5
c_2 KL divergence loss coefficient	0.01
c_3 Entropy loss coefficient	0.001
η_{c_3} Entropy coefficient decay factor	0.98

effectiveness of the model's learning process. The following methods are implemented:

- Parallel environments
- Shared networks for the critic and actor
- Generalized Advantage Estimation
- Curriculum learning
- Mini-batch update
- KL divergence penalty in the loss function
- KL divergence threshold-based learning rate decaying
- Entropy bonus in the loss function
- Normalized advantages
- Orthogonal initialization of weights and constant initialization of biases
- Gradient clipping $\pm 4\sqrt{v}$ where v is the running estimate of the gradient's second moment before clipping, similar to [5]
- Entropy coefficient annealing as emphasized in [31]
- PPO's clip range annealing

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[2] O. Vinyals, "StarCraft II: A new challenge for reinforcement learning," 2017, *arXiv:1708.04782*.

[3] P.-A. Andersen, M. Goodwin, and O.-C. Granmo, "Deep RTS: A game environment for deep reinforcement learning in real-time strategy games," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Aug. 2018, pp. 1–8.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, and P. Georgiev, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 7782.

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

[6] S. Huang, S. Ontañón, C. Bamford, and L. Grela, "Gym-μRTS: Toward affordable full game real-time strategy games research with deep reinforcement learning," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2021, pp. 1–8.

[7] V. G. Goecks, N. Waytowich, D. E. Asher, S. Jun Park, M. Mittrick, J. Richardson, M. Vindiola, A. Logie, M. Dennison, T. Trout, P. Narayanan, and A. Kott, "On games and simulators as a platform for development of artificial intelligence for command and control," *J. Defense Model. Simul., Appl., Methodol., Technol.*, vol. 20, no. 4, pp. 495–508, Oct. 2023.

[8] D. Barone, J. Wickboldt, M. Cavalcanti, D. Moura, J. Tesolin, A. Demori, J. Anjos, L. Silva de Carvalho, J. Gomes, and E. Pignaton de Freitas, "Integrating a multi-agent system simulator and a network emulator to realistically exercise military network scenarios," in *Proc. 13th Int. Conf. Simulation Modeling Methodologies, Technol. Appl.*, 2023, pp. 194–201.

[9] J. Boron and C. Darken, "Developing combat behavior through reinforcement learning in wargames and simulations," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2020, pp. 728–731.

[10] B. Toghiani-Rizi, F. Kamrani, L. J. Luotsinen, and L. Gisslén, "Evaluating deep reinforcement learning for computer generated forces in ground combat simulation," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 3433–3438.

[11] S. Yu, W. Zhu, and Y. Wang, "Research on wargame decision-making method based on multi-agent deep deterministic policy gradient," *Appl. Sci.*, vol. 13, no. 7, p. 4569, Apr. 2023.

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

[13] Q. Fu, C.-L. Fan, Y. Song, and X.-K. Guo, "Alpha C2—An intelligent air defense commander independent of human decision-making," *IEEE Access*, vol. 8, pp. 87504–87516, 2020.

[14] V. Chmyr and N. Bhinder, "AI in the higher military institutions: Challenges and perspectives for military engineering training," *Rupkatha J. Interdiscipl. Stud. Humanities*, vol. 15, no. 4, p. 1, Dec. 2023.

[15] L. Liu, N. Gurney, K. McCullough, and V. Ustun, "Graph neural network based behavior prediction to support multi-agent reinforcement learning in military training simulations," in *Proc. Winter Simulation Conf. (WSC)*, Dec. 2021, pp. 1–12.

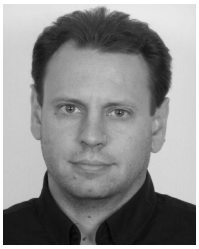
[16] A. Basak, E. G. Zaroukian, K. Corder, R. Fernandez, C. D. Hsu, P. K. Sharma, N. R. Waytowich, and D. E. Asher, "Utility of doctrine with multi-agent RL for military engagements," in *Proc. Artif. Intell. Mach. Learn. Multi-Domain Oper. Appl. IV*, Jun. 2022, pp. 609–628.

[17] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The StarCraft multi-agent challenge," 2019, *arXiv:1902.04043*.
[18] H. Zhou, Y. Mao, and X. Guo, "An improved multi-objective particle swarm optimization-based hybrid intelligent algorithm for index screening of underwater manned/unmanned cooperative system of systems architecture evaluation," *Mathematics*, vol. 11, no. 20, p. 4389, Oct. 2023.
[19] M. Sommer, M. Rüegsegger, O. Szehr, and G. Del Rio, "Deep self-optimizing artificial intelligence for tactical analysis, training and optimization," in *Proc. NATO NMSG Symp., Towards Training Decis. Support Complex Multi-Domain Oper.*, 2021, p. 1.
[20] L. T. Doll, M. Behm, J. Brendecke, and D. Kallfass, "From the game map to the battlefield–using deepmind's advanced alphastar techniques to support military decision-makers," in *Proc. Towards Training Decis. Support Complex Multi-Domain Oper., NATO Modeling Simulation Group (NMSG) Syposium*, Amsterdam, The Netherlands, 2021, pp. 1–14.
[21] L. Espeholt, "IMPALA: Scalable distributed deeP-RL with importance weighted actor-learner architectures," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1407–1416.
[22] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.
[23] M. Hua. (2022). *Pycmo*. [Online]. Available: https://github.com/duyminh1998/pycmo
[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, Lille, France, 2015, pp. 1889–1897.
[25] O. Delalleau, M. Peter, E. Alonso, and A. Logut, "Discrete and continuous action representation for practical RL in video games," 2019, *arXiv:1912.11077*.
[26] E. Wei, D. Wicke, and S. Luke, "Hierarchical approaches for reinforcement learning in parameterized action space," 2018, *arXiv:1810.09656*.
[27] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Jan. 1990.
[28] A. V. Clemente, H. N. Castejón, and A. Chandra, "Efficient parallel methods for deep reinforcement learning," 2017, *arXiv:1705.04862*.
[29] Y. Bengio, J. Louradour, and R. Collobert, "Curriculum learning," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2009, pp. 41–48.
[30] R. da Rosa, M. A. Wehrmeister, T. Brito, J. L. Lima, and A. I. P. N. Pereira, "Honeycomb map: A bioinspired topological map for indoor search and rescue unmanned aerial vehicles," *Sensors*, vol. 20, no. 3, p. 907, Feb. 2020.
[31] Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans, "Understanding the impact of entropy on policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 151–160.

ADONISZ DIMITRIU was born in Hungary, in 1996. He received the B.Sc. and M.Sc. degrees in electrical engineering from Budapest University of Technology and Economics, Hungary, in 2021 and 2023, respectively. He is currently an ML Engineer focusing on reinforcement learning. His research interest includes artificial intelligence-based multi-agent systems.

TAMÁS V. MICHALETZKY was born in Hungary, in 1997. He received the B.Sc. degree in mathematics and the M.Sc. degree in theoretical mathematics from Eötvös Loránd University, Hungary, in 2019 and 2022, respectively. He was a Research Trainee with the Institute for Computer Science and Control, in 2021, and a Data Scientist at an IT Company, from 2022 to 2023. He is currently an ML Engineer focusing on reinforcement learning.

VIKTOR REMELI was born in Belgrade, Yugoslavia, in 1987. He received the degree from the Faculty of Information and Communication Technology, University of Malta, in 2015. He worked for ten years in the software industry and five years as an Assistant Research Fellow with the Department of Automotive Technologies, Budapest University of Technology and Economics. He is currently the AI Research Team Leader with the TECHTRA-Technology Transfer Institute, Hungary. His research interests include deep learning-based environment perception methods and their verification, as well as solutions to complex and cooperative planning problems in robotic environments.



VIKTOR R. TIHANYI was born in Hungary, in 1981. He received the degree in electrical engineering from the Faculty of Electric Machines and Drives, Budapest University of Technology and Economics, in 2005, the B.Sc. degree in mechanical engineering from the Vehicle Technology Faculty, University of Óbuda, in 2014, and the Ph.D. degree, in 2012. He has been with Hyundai Technology Center Hungary for five years, since 2008. In 2013, he switched to the Automotive Sector, Knorr-Bremse Fékrendszerek Kft., as the Project Leader and the Team Leader of electromobility and autonomous vehicle-related projects, until 2019. He worked for three years at the ZalaZONE proving ground as the Team Leader of research and innovation activities. Besides his industrial employment, he has also been with the Department of Automotive Technologies, Budapest University of Technology and Economics, as the Research Leader of autonomous vehicle-related research projects, since 2016, as an Associate Professor. He is the Deputy CEO of the TECHTRA-Technology Transfer Institute, Hungary, and the Leader of the institute's research and development division.