Unreal-MAP: Unreal-Engine-Based General Platform for Multi-agent Reinforcement Learning

Tianyi Hu[1,2,3], **Qingxu Fu**[4], **Zhiqiang Pu**[1,2,3*], **Yuan Wang**[1,2,3], **Tenghai Qiu**[1,2]

[1]Institute of Automation, Chinese Academy of Sciences
[2]National Key Laboratory of Cognition and Decision Intelligence for Complex Systems
[3]School of Artificial Intelligence, University of Chinese Academy of Sciences
[4]Alibaba (China) Co., Ltd., Beijing

Abstract

In this paper, we propose *Unreal Multi-Agent Playground* (Unreal-MAP), an MARL general platform based on the Unreal-Engine (UE). Unreal-MAP allows users to freely create multi-agent tasks using the vast visual and physical resources available in the UE community, and deploy state-of-the-art (SOTA) MARL algorithms within them. Unreal-MAP is user-friendly in terms of deployment, modification, and visualization, and all its components are open-source. We also develop an experimental framework compatible with algorithms ranging from rule-based to learning-based provided by third-party frameworks. Lastly, we deploy several SOTA algorithms in example tasks developed via Unreal-MAP, and conduct corresponding experimental analyses including a sim2real demo. We believe Unreal-MAP can play an important role in the MARL field by closely integrating existing algorithms with user-customized tasks, thus advancing the field of MARL.

Code — https://github.com/binary-husky/unreal-map
HMAP — https://github.com/binary-husky/hmp2g
Extended version — https://arxiv.org/pdf/2503.15947

Introduction

Multi-agent reinforcement learning (MARL) has demonstrated remarkable potential in many practical fields, including swarm robotic control (Kalashnikov et al. 2018; Chen, Chang, and Zhang 2020), autonomous vehicles (Peng et al. 2021b), and video games (Vinyals et al. 2019). There are many classical algorithms that have emerged in the field of MARL, such as QMIX (Rashid et al. 2020b), QPLEX (Wang et al. 2020), MAPPO (Yu et al. 2022) and HAPPO (Zhong et al. 2024). The success of these algorithms could not be achieved without the support of numerous well-designed simulation environments. These environments provide abundant simulated data and standardized benchmarks for deploying and comparing algorithms, thereby driving the development of more advanced algorithms.

However, existing simulation environments are predominantly domain-specific, which restricts their capacity to accommodate diverse user needs and real-world applications. When considering rapid deployment in practical applications, current popular environments cannot be freely modified. For every specific need, building domain-specific environments from scratch one by one is inefficient and costly. Researchers need to balance the realism of scenarios, the practicality of code, and compatibility with state-of-the-art (SOTA) algorithms (Oroojlooy and Hajinezhad 2023). Therefore, if an MARL-compatible platform can be developed that allows users to build on its resources and ready-made functions, it will significantly reduce development costs and greatly promote the practical application of MARL.

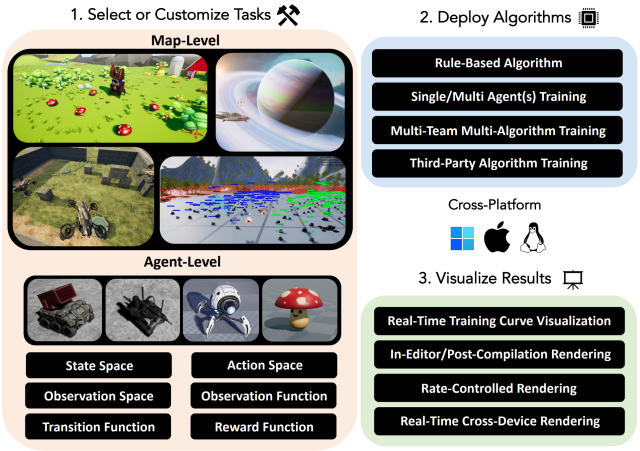

Figure 1: The research workflow for using Unreal-MAP. For novice users, Unreal-MAP provides direct access to built-in tasks, and offers comprehensive algorithm deployment functions and visualization capabilities. For advanced users, Unreal-MAP enables the modification of built-in tasks or the creation of new tasks to test research ideas.

How can such a platform be built? Modern game engines provide a possible technical foundation. Game engines are software frameworks originally designed to simplify the development of video games (Boyd and Barbosa 2017). Over years of evolution, modern game engines have established vast development communities, along with rich rendering resources and physics engine materials (Wheeler

*Corresponding author. Email: zhiqiang.pu@ia.ac.cn.

2023). These potential resources could greatly facilitate the application deployment within the MARL domain. Moreover, game engines are closely linked with the rapidly developing generative AI, which has the potential to quickly transform user needs into real products. Some works have utilized simulation data from game engines for the training of generative AI (Lu et al. 2024), whereas others have established a feedback loop, incorporating these trained generative models into game engine plugins to facilitate scene creation (NVIDIA 2024). This suggests that combining game engines with MARL could have a very promising future.

However, there remains a significant gap between the game development community and the MARL community. Although some efforts, such as Unity-ML Agents (Juliani 2018) and URLT (Sapio and Ratini 2022), have built frameworks for freely constructing RL training scenarios using game engines, they still face issues in training efficiency and scaling complexity (Kaup et al. 2024). In response to the aforementioned issues, we propose Unreal-MAP, an MARL general platform based on the Unreal Engine. Compared to existing general platforms, Unreal-MAP possesses the following main features: **(1) Fully Open-Source and Easily Modifiable**, Unreal-MAP utilizes a layered design, and all components are open-sourced. Users can easily modify all elements of an MARL task by focusing only on the high-level layers. **(2) Optimized Specifically for MARL**, the underlying engine of Unreal-MAP has been optimized to enhance efficiency in large-scale agent training. This optimization allows users to develop simulations with heterogeneous, large-scale, multi-team settings that showcase distinctive multi-agent features through Unreal-MAP. **(3) Parallel Multi-Process Execution and Controllable Single-Process Time Flow**, Unreal-MAP supports the parallel execution of multiple simulation processes as well as the adjustment of the simulation speed in a single process. Users can accelerate simulations to speed up training or decelerate simulations for detailed slow-motion analysis. Compared with existing MARL environments, Unreal-MAP not only provides the advantage of freely customizing highly realistic scenarios, but also includes features such as cross-device real-time rendering. A detailed comparison with current representative MARL environments and general RL platforms can be found in Table 1.

To fully utilize the capabilities of Unreal-MAP, we also develop an MARL experimental framework known as the Hybrid Multi-Agent Playground (HMAP). This framework includes implementations of rule-based algorithms, built-in learning-based algorithms, and algorithms from third-party frameworks such as PyMARL2 (Hu et al. 2021) and HARL (Zhong et al. 2024). By leveraging Unreal-MAP and HMAP, users can rapidly customize environments and deploy algorithms, validate new research ideas, and apply them in practical scenarios. The overview of the research workflow for using Unreal-MAP is depicted in Figure 1 and more details can be found in Appendix E.

The contributions of this work are summarized as follows: firstly, an MARL general platform based on the Unreal Engine; secondly, a modular MARL experimental framework; thirdly, a collection of highly extensible example scenarios based on Unreal-MAP and related experimental analyses. We believe Unreal-MAP can serve as a comprehensive tool to advance the development of MARL and ultimately facilitate their application in real-world scenarios.

Related Work

Simulation Environments for MARL. Existing MARL environments can be broadly divided into *domain-specific environments* and *environment suites*. The former includes a series of tasks designed around the same domain, which usually share a common genre and similar benchmark metrics. Notable examples of these works include Multi-Agent Particle Environment (MPE) (Mordatch and Abbeel 2017), StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al. 2019) and Google Research Football (GRF) (Kurach et al. 2020). Due to architectural constraints, domain-specific environments cannot be freely modified according to user needs. For instance, a StarCraft-based SMAC scenario can never be reconfigured into soccer-style tasks. On the other hand, *environment suites* consist of sets of environments packaged together, commonly used to more conveniently benchmark the performance of algorithms. Some works also redesign built-in environments to enhance training speed. Typical works in this category include PettingZoo (Terry et al. 2021) and JaxMARL (Rutherford et al. 2024). Nevertheless, these suites are just collections of domain-specific environments and thus lack flexible modification capabilities. More details of related MARL environments can be found in Appendix D.

General Platforms for RL. These kinds of works enable users to create environments with specified visual or physical complexity and deploy RL algorithms. GodotRL (Beeching et al. 2021) is a framework for building RL scenarios using the Godot engine, but its training tasks are primarily focused on pure game scenarios and are not compatible with the latest MARL algorithms. Unity-ML Agents (Juliani 2018) is a mature general platform built on the Unity engine. It allows users to develop new scenes using the Unity editor and train them via RL algorithm libraries. However, its limited open-source implementation hinders further customization (Wheeler 2023), and there are still issues in scaling complexity and simulation fidelity (Kaup et al. 2024). Compared to Godot engine, Unreal Engine (UE) offers richer community resources and more powerful physics simulation capabilities; compared to Unity, UE offers full open-source access and lower learning curves[1] (Boyd 2017), is more conducive to the development of a general platform. URLT (Sapio and Ratini 2022) is an RL general platform based on UE, but its algorithm relies on Blueprint and is difficult to integrate with existing libraries. Moreover, lacking controllable time flow training, making it inapplicable to large-scale simulations. Currently, there is a lack of a powerful, MARL-targeted general platform.

[1]The UE editor employs the graphical programming language Blueprints, offering a lower learning curve than its native C++ library, even for those familiar with C++ (Boyd 2017).

Related Work	*Heterogeneous Support*	*Large-Scale Support*	*Multi-Team Support*	*Mixed-Game Support*	*3D Physics Engine*	*Fully Open Source*	*Free task Construction*	*Controllable Sim Speed*	*Render During Training*	*SOTA MARL Compat*
MPE (Mordatch and Abbeel 2017)	✓	-	-	✓	-	✓	-	-	-	✓
MAgent (Zheng et al. 2018)	✓	✓	-	✓	-	✓	-	-	-	✓
Hanabi (Bard et al. 2020)	-	-	-	-	-	✓	-	-	-	✓
NeuralMMO (Suarez et al. 2021)	✓	✓	-	✓	-	-	-	-	✓	✓
GoBigger (Zhang et al. 2022)	-	✓	✓	✓	-	-	-	-	-	✓
JaxMARL (Rutherford et al. 2024)	✓	✓	-	✓	-	✓	-	-	-	✓
GRF (Kurach et al. 2020)	✓	-	-	-	✓	✓	-	-	-	✓
SMAC (Samvelyan et al. 2019)	✓	-	-	✓	✓	-	-	-	-	✓
SMACv2 (Ellis et al. 2022)	✓	-	-	✓	✓	-	-	-	-	✓
Hide-and-Seek (Baker et al. 2019)	-	-	-	-	✓	✓	-	-	-	✓
HoK3v3 (Liu et al. 2023)	✓	-	-	-	✓	-	-	-	-	✓
MAMuJoCo (Peng et al. 2021a)	✓	✓	-	-	✓	✓	-	-	-	✓
Marathon (Booth and Booth 2019)	✓	-	-	-	✓	-	-	-	✓	-
GodotRL (Beeching et al. 2021)	✓	-	-	✓	-	✓	✓	-	✓	-
URLT (Sapio and Ratini 2022)	-	-	-	-	-	✓	✓	-	-	-
Unity-ML Agents (Juliani 2018)	✓	-	-	✓	-	-	✓	✓	✓	-
Unreal-MAP (ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison between Unreal-MAP and other representative MARL environments and RL general platforms. Red shaded areas represent general platforms. More details of related works can be found in Appendix D.

Background

To accommodate various interaction relationships among multi-agent and multi-team scenarios (Fu et al. 2024), we use Partially Observable Markov Game (POMG) (Littman 1994; Gronauer and Diepold 2022) to model the MARL problem. A POMG can be represented by an 8-tuple $\langle N, \{S^i\}_{i\in N}, \{O^i\}_{i\in N}, \{\Omega^i\}_{i\in N}, \{A^i\}_{i\in N}, \{\mathcal{T}^i\}_{i\in N}, r, \gamma\rangle$. N is the set of all agents, $\{S^i\}_{i\in N}$ is the global state space which can be factored as $\{S^i\}_{i\in N} = \times_{i\in N} S^{(i)} \times S^E$, where $S^{(i)}$ is the state space of an agent i, and S^E is the environmental state space, corresponding to all the non-agent entities. $\{O^i\}_{i\in N} = \times_{i\in N} O^{(i)}$ is the joint observation space and $\{\Omega^i\}_{i\in N}$ is the set of observation functions. Similarly, $\{A^i\}_{i\in N}$ is the joint action space of all agents. $\{\mathcal{T}^i\}_{i\in N}$ is the collection of all agents' transitions and the environmental transition. Finally, γ is the discount factor and $r : \{S^i\}_{i\in N} \times \{A^i\}_{i\in N} \times N \to \mathbb{R}$ is the agent-level reward function.

We define *team* as a collection of agents, which all share the same overall goal in a purely cooperative form. Agents within the same team aim to find an optimal joint policy that maximizes the cumulative reward for the whole team. Denoting the joint policy of a certain team $A \subseteq N$ as $\bar{\pi}_A$, the optimal policy $\bar{\pi}^*_A$ can be represented as:

$$\bar{\pi}^*_A = \arg\max_{\bar{\pi}_A} \mathbb{E}_{\bar{\pi}_A}\left[\sum_{k=0}^{\infty}\gamma^k \sum_{i\in A} r^i_{t+k} \mid \bar{s}_t = \bar{s}\right], \qquad (1)$$

where $\bar{s}$ is the initial global state, $\gamma^k \sum_{i\in A} r^i_{t+k}$ is the discounted return of team A, r^i_{t+k} is the reward of an agent $i \in A$ at timestep $t + k$.

Unreal-MAP

Basic Concepts

Multi-agent simulation can demonstrate great diversity in different domains. We introduce several new concepts that align with human intuition as well as the requirements of multi-agent simulation.

Agents and **Teams:** Agents are the basic decision-making units in the environments. Unreal-MAP introduces a new concept *team* to distinguish agents with different goals. Unreal-MAP supports numbers of teams, where teams may engage in competition or cooperation. Each team possesses its own independent goal and is equipped with a separate learning-based (or rule-based) algorithm.

Entities: Entities are objects in simulation that do not make decisions but still have important functionality. For instance *a street lamp* or *a dynamic obstacle*. A shared characteristic of these objects is that they must be removed or reinitialized when an episode ends or a new episode starts.

Tasks and **Scenarios:** Tasks correspond to POMGs defined in Section Background. The properties of tasks in Unreal-MAP include the types and numbers of agents, their team affiliations, as well as each agent's state space, action space, etc. A scenario is a *"meta-task"* that can give rise to a series of tasks. Tasks generated from the same scenario share the same genre and similar objectives.

Maps: Maps in Unreal-MAP determine where the task takes place. A map can be *a small room*, or *a city full of buildings*. It is a great advantage that Unreal-MAP decouples the concept of tasks and maps, as users can conveniently deploy a task in new maps (as long as the agent has the appropriate size and a suitable position initialization function).

Events: We define an event system to simplify the reward

crafting procedure. For instance, an event will be generated when an agent is destroyed or an episode is ended. When it is time to compute next-step reward, these events will provide convenient reference.

Utilizing Unreal-MAP to customize tasks

Unreal-MAP employs a hierarchical five-layer architecture, where each layer builds upon the previous one. From bottom to top, the five layers are: *native layer*, *specification layer*, *base class layer*, *advanced module layer*, and *interface layer*. **Users only need to focus on the *advanced module layer* and the *interface layer*.** In most cases, modifying the *interface layer* is sufficient to alter all elements of tasks.

Fundamental Layers. Figure 2 shows the internal architecture of Unreal-MAP. Specifically, the *native layer* includes assets from the Unreal community and the Unreal Engine, some part of which have been optimized for MARL compatibility. The *specification layer* consists of Unreal-MAP's underlying systems and programming specifications, all implemented in C++. The *base class layer* includes all basic classes implemented using Blueprints. These three layers, also known as the fundamental layers, form the foundation of Unreal-MAP.

User Operation Layers. The top two layers of Unreal-MAP are user operation layers. The *advanced module layer*, based on Blueprints, allows for the modification of agents' physical properties such as appearances and kinematics, thereby enabling the development of various agents. This layer also facilitates the development of environmental entities and maps. The top layer is the *interface layer*, implemented in Python and compliant with the gym standard. It supports customizable observations and reward functions, and allows for the selection of maps and agents. Users can easily customize all elements of POMG tasks through simple operations via top layers. More details about the architecture of Unreal-MAP and how the elements can be customized can be found in Appendix E.

Other Features of Unreal-MAP

Computational efficiency. Numerous modifications have been made to the underlying engine to adapt it for efficient MARL training. These include optimizations within the simulation engine and enhancements in the communication between the simulator and the algorithm side (details are provided in Appendix E.4). In practical training, Unreal-MAP also supports a non-render training mode without rendering frame computation.

Controllable simulation time flow. Unreal-MAP optimizes the time flow control mechanism in Unreal Engine (details are provided in Appendix E.3). Users can easily modify the time dilation factor to adjust the ratio of simulated time flow and real time flow. The ability to control the time flow offers numerous benefits. On one hand, users can accelerate the simulation time for rapid training or decelerate for debugging. On the other hand, since adjusting the speed of simulation does not influence memory resources, users can make fuller use of computational resources by adjusting the time dilation factor, with more detailed information available in Section Experiments.

Compatibility with multiple systems and computational backends. Unreal-MAP is natively compatible with creating tasks and deploying algorithms on Windows, Linux, and MacOS. It supports training on pure CPU setups as well as on hybrid CPU/GPU configurations. Furthermore, Unreal-MAP supports cross-device real-time rendering[2], allowing users to conduct multi-process training on a Linux server while performing real-time rendering of specific processes on a Windows host.

HMAP

To facilitate the deployment of algorithms for Unreal-MAP, we also develop an experimental framework HMAP. HMAP is a multi-agent experimental framework with decoupled *Task-Core-Algorithm* components. Currently, HMAP not only integrates Unreal-MAP's tasks, but also supports other MARL environments such as SMAC (Samvelyan et al. 2019) and MPE (Mordatch and Abbeel 2017). On the algorithm side, HMAP also supports a wide range of algorithms. This includes rule-based algorithms (most of them are built-in policies for Unreal-MAP example tasks), single-agent RL algorithms like DQN (Mnih et al. 2015) and SAC (Haarnoja et al. 2018), as well as MARL algorithms such as MAPPO (Yu et al. 2022) and HAPPO (Zhong et al. 2024). Furthermore, HMAP is compatible with third-party frameworks, supporting all algorithms from PyMARL2 (Hu et al. 2021) and HARL (Zhong et al. 2024).

The unique feature of HMAP is its support for multi-team training. By thoroughly decoupling algorithms from tasks, HMAP employs its core as a *"glue module"*, enabling any algorithm module to control teams within any task module. The highly modular design presents three key benefits. Firstly, it enables modification of built-in policies in tasks within Python-based algorithm modules, which can significantly reduce the workload of building non-learning-based policy[3] on the UE side. Secondly, it enables teams controlled by multiple algorithms to interact within a same task, facilitating co-training of algorithms from different frameworks under the same task. Thirdly, it is user-friendly, **as all experimental configurations based on HMAP can be implemented through a single JSON file.** After completing the configuration, users can initiate the training task with just one line of code. More details of HMAP can be found in Appendix F.

Example Scenarios and Tasks

Unreal-MAP includes a variety of basic scenarios for multi-agent systems, each of which is extensible and can be used to create numerous tasks. This section describes 4 example scenarios, which are used to develop 15 tasks applied in Section Experiments. We use these example scenarios to demonstrate that Unreal-MAP can be used to construct tasks with distinct multi-agent characteristics. These characteristics include heterogeneity, large-scale, multi-team,

[2]Remotely connecting to a non-render client running inside a server via network, and rendering the ongoing training process locally via TCP&UDP.

[3]For example, the policy of a Non-Player Character (NPC).

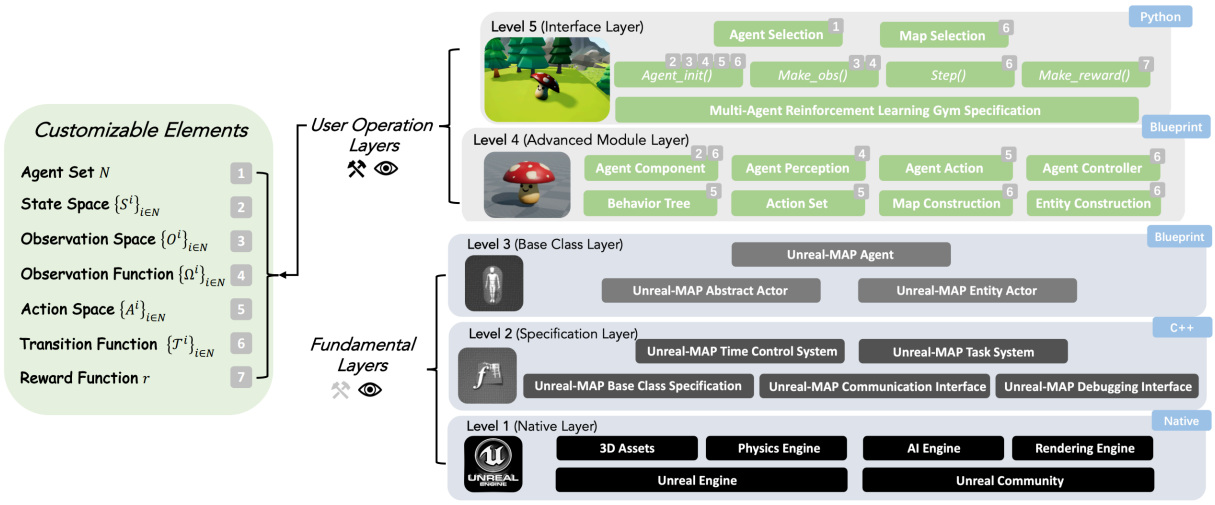

Figure 2: Architecture of Unreal-MAP. Unreal-MAP employs a hierarchical, five-layered architecture, all of which are open sourced. Users can modify all elements within POMG by configuring parameters through the Python-based *interface layer*. For more advanced development requirements, users can conveniently adjust scenario elements using Blueprint through the *advanced module layer*.

sparse team rewards, and multi-agent games. More details of these example scenarios can be referred to Appendix G.

Metal Clash - designed for heterogeneous and large-scale tasks. It involves an SMAC-style competition between two teams of agents. Each team can be controlled by rule-based or learning-based algorithms. This scenario provides three types of basic agents: missile cars, laser cars and support drones. The properties of each basic agent, such as max-speed and HP can be easily modified, thus creating a variety of heterogeneous agent types beyond the original three. The number and types of agents in each team can be freely changed, altering the features and difficulty of the tasks.

Monster Crisis - designed for sparse team reward tasks. This is a village-style scenario where several mushroom agents need to resist the invasion of a monster. The entire team receives a positive reward only if they kill the monster, and there are no rewards or penalties in other cases. Users can adjust the difficulty by modifying the monster's HP and the number of agents.

Flag Capture - designed for multi-team gaming tasks. It involves several robot teams gaming to capture a flag. The closest robot can pick up the flag, and their teammates must defend it from other teams. The team with longest flag possession wins. The number of agents and teams can be changed to modify task features and difficulty.

Navigation Game - designed for heterogeneous and two-team zero-sum gaming tasks. It includes two landmarks, a keeper team, and a navigator team. Although they cannot attack each other, the ground keeper can drive away the air navigator, while the ground navigator can drive away the ground keeper. If the air navigator stays over any landmark for a certain period, the navigator team wins. The rewards for the two teams are zero-sum. The task difficulty can be changed by adjusting team sizes.

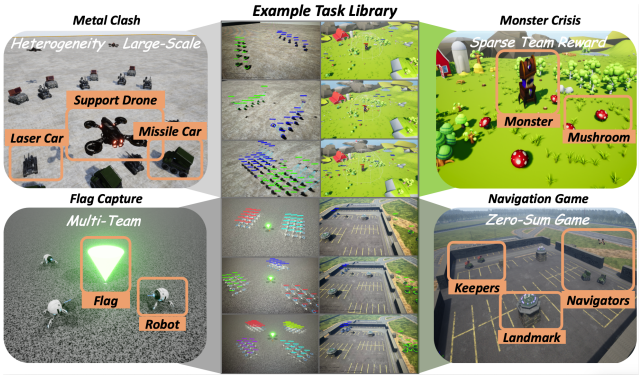

Figure 3: Example scenarios and tasks of Unreal-MAP. Users can develop new scenarios using Unreal-MAP, and create a variety of MARL tasks by adjusting properties such as the number, types, and teams of agents.

Experiments

The experimental section is divided into three main parts: First, we test several SOTA MARL algorithms on example tasks. This research demonstrates how Unreal-MAP effectively integrates with SOTA algorithms and demonstrates its capability to simulate tasks with diverse features. Second, we measure resource consumption and training efficiency. We demonstrate the results compared to other works, and show how Unreal-MAP benefits from its multi-processing and adjustable time flow features. Third, physical experiments validate Unreal-MAP's physics fidelity and sim2real transfer potential.

Performance in Example Tasks. We develop 15 example tasks based on 4 scenarios from Unreal-MAP, as detailed in Table 2. Based on HMAP, we deploy 7 SOTA MARL algorithms on all tasks, including the actor-critic-

Example Task	MARL Agents	Other Entities	Features	Remark
metal_clash_5lc_5mc	5 LC, 5 MC	5 LC, 5 MC	*Heterogeneous*	-
metal_clash_het_10	4 LC, 4 MC, 2 SD	4 LC, 4 MC, 2 SD	*Heterogeneous*	-
metal_clash_het_8_vs_10	4 LC, 2 MC, 2 SD	4 LC, 4 MC, 2 SD	*Heterogeneous*	-
metal_clash_hom_50	50 LC	50 LC	*Large-Scale*	-
metal_clash_het_50	20 LC, 20 MC, 10 SD	20 LC, 20 MC, 10 SD	*Heterogeneous + Large-Scale*	-
metal_clash_het_100	40 LC, 40 MC, 20 SD	40 LC, 40 MC, 20 SD	*Heterogeneous + Large-Scale*	-
monster_crisis_easy	8 Mushroom	1 Monster	*Sparse Team Reward*	HP of Monster is 400
monster_crisis_medium	8 Mushroom	1 Monster	*Sparse Team Reward*	HP of Monster is 600
monster_crisis_hard	8 Mushroom	1 Monster	*Sparse Team Reward*	HP of Monster is 800
flag_capture_1script	15 Robot	15 Robot	*Zero-Sum Game*	-
flag_capture_2scripts	15 Robot	2 * [15 Robot]	*Multi-Team*	-
flag_capture_2scripts_hard	10 Robot	2 * [15 Robot]	*Multi-Team*	-
navigation_game_5_vs_2	3 GN, 2 AN	2 GK	*Heterogeneous + Game*	-
navigation_game_4_vs_2	2 GN, 2 AN	2 GK	*Heterogeneous + Game*	-
navigation_game_3_vs_2	2 GN, 1 AN	2 GK	*Heterogeneous + Game*	-

Table 2: Description of example tasks in the experiments (LC: Laser-Car, MC: Missile-Car, SD: Support-Drone, GN: Ground-Navigator, GK: Ground-Keeper, AN: Air-Navigator).

Metric	**Ours**	SMAC	MPE	Unity-ML	JAXMARL
TPS ($\times 10^3$) ↑	1.04	1.21	1.62	**10.2**	**200-2700**
FPS ($\times 10^3$) ↑	**1340**	9.67	1.62	10.2	200-2700
CPU (%) ↓	8.9	**4.6**	9.5	25.6	–
Mem (GB) ↓	13.6	33.5	**3.9**	10.2	132
GPU (GB) ↓	2.7	2.9	**1.5**	5.6	<24

Table 3: Efficiency and resource consumption metrics of Unreal-MAP compared to other works.

based algorithms as MAPPO (Yu et al. 2022), HATRPO and HAPPO (Zhong et al. 2024), as well as the value-based algorithms as QMIX (Rashid et al. 2020b), QTRAN (Son et al. 2019), QPLEX (Wang et al. 2020) and WQMIX (Rashid et al. 2020a). To ensure a fair comparison, the main network and hyperparameters of each algorithm are standardized (details in Appendix J.1).

Figure 4 shows the results (with lines representing mean values and shaded areas indicating 95% confidence intervals), demonstrating that Unreal-MAP can develop MARL-compatible tasks. By comparing the test results of different tasks developed from the same scenario, it is evident that changing ① the property of individual entities or agents ② types of agents ③ number of agents ④ number of agent teams can effectively alter the features and difficulty of the tasks. It is worth mentioning that we DO NOT intend to develop the example tasks as benchmarks, but rather use them to show that based on Unreal-MAP, it is possible to develop extensible MARL-compatible tasks. Moreover, we conducted a performance analysis of the algorithm, with detailed findings presented in Appendix J.

Efficiency and Computational Consumption. We conduct experiments on the efficiency and resource consumption metrics. The efficiency metrics adopted are TPS/FPS i.e., the number of *virtual (Timesteps/Frames)* Per *real Second*. The resource consumption metrics include CPU utilization, memory occupancy, and GPU memory occupancy.

First, we compare the above metrics of Unreal-MAP with several related works, including two MARL environments (SMAC & MPE), a general RL platform (Unity-ML Agents), and a GPU-accelerated MARL environment (JAXMARL). To ensure a fair comparison, we select MARL tasks with similar numbers of agents (approximately 20) and complexity whenever possible. The experimental results are shown in Table 3, with details provided in Appendix H. The results indicate that Unreal-MAP demonstrates acceptable resource utilization and achieves TPS comparable to common environments. Notably, each timestep in Unreal-MAP undergoes 1280 frames of calculations for environmental dynamics (details in Appendix E.3 and J.1). This enables the speed of simulation physics frame calculation to reach the 1M level, resulting in the highest FPS among the compared works.

Moreover, Unreal-MAP regulates efficiency and resource usage via two parameters: process count and time dilation factor. Figure 5 demonstrates that increasing time dilation factor only increases CPU utilization, with almost no effect on memory and GPU memory. In cases with limited memory resources, users can consider lowering the process count and increase time dilation factor, or conversely in CPU-constrained situations. **Combining Unreal-MAP's inherent cross-platform compatibility, it is resource-friendly across various computing devices.**

Physical Experiment. We conduct this experiment to demonstrate the potential of Unreal-MAP in bridging the sim-to-real gap. First, we establish a real-world setup consisting of a motion capture system, a communication system, several autonomous UGVs and UAVs, and a number of physical entities. Subsequently, we construct its digital twin via Unreal-MAP (based on the *navigation game* scenario), where the entities are proportionally replicated from the physical setup, and the kinematics of the unmanned units are also recreated. Finally, we develop an *algorithm-Unreal-MAP-hardware* framework, with details presented in Appendix I.

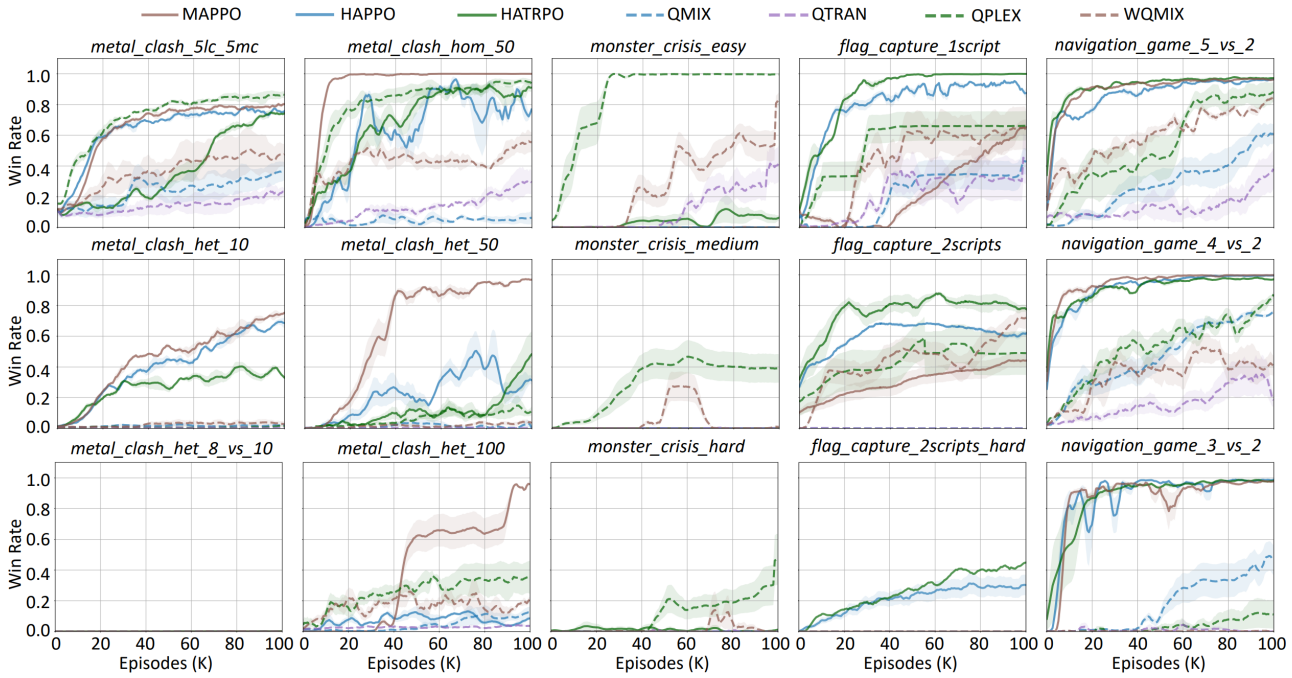

Figure 4: The comparison of test win rate for all tested algorithms across 15 example tasks. The shadowed area depicts the 95% confidence interval.

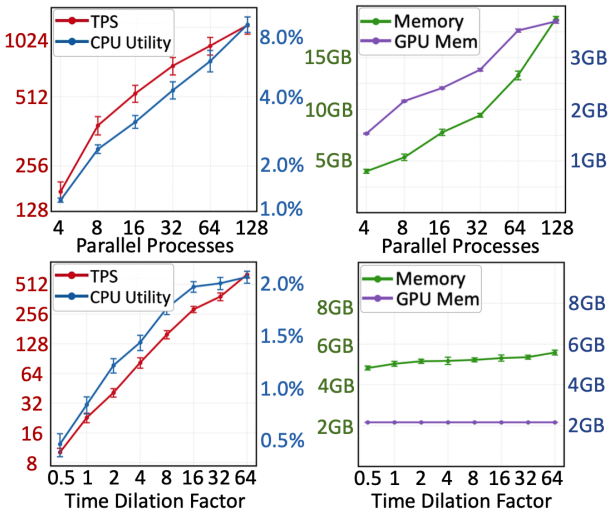

Figure 5: The impact of the number of parallel processes and the time dilation factor in Unreal-MAP on simulation efficiency and computational resource consumption.

We train the policies in the virtual environment and then deploy them to the physical setup. Figure 6 presents snapshots from both the sim/real scenarios. The results show that the agent trajectories generally align well. In virtual testing, we conduct 512 episodes with a 94.1% success rate. Real-world validation across 15 episodes achieves a comparable 93.3% success rate. The experimental results indicate that the whole system can successfully replicate the multi-agent policies from the virtual environment within the real-world.

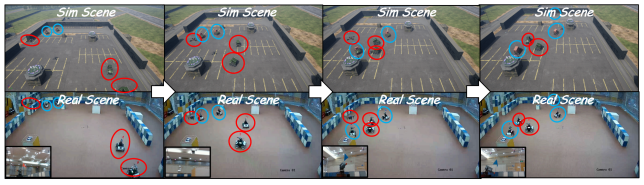

Figure 6: Snapshots from Unreal-MAP-simulated and real-world scenarios. The top four subfigures shows snapshots of multiple agents deploying well-trained policies only in virtual scenarios. The bottom four subfigures shows deployed policies in real-world scenarios at the corresponding timesteps. The red-circled agents are navigators, and the blue-circled agents are keepers.

Conclusion

In this paper, we propose Unreal-MAP, a powerful MARL general platform. The scenarios developed by Unreal-MAP exhibit high extensibility and realism, and its layered design enables users to quickly build MARL training tasks. Using Unreal-MAP, we develop a series of example tasks incorporating features such as heterogeneity, large-scale, and multi-team interactions. Furthermore, we develop HMAP, a comprehensive multi-agent experimental framework. Through the integration of Unreal-MAP and HMAP, we conduct validation experiments and comparative studies. Finally, we replicate a task in a real-world setting, demonstrating Unreal-MAP's potential to bridge virtual algorithms with real-world applications. The limitations and future work of Unreal-MAP are discussed in Appendix B.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62322316, the Beijing Nova Program under Grant 20220484077 and 20230484435, the National Natural Science Foundation of China under Grant No. 62503472, and the Open Fund of National Key Laboratory of Information Systems Engineering (No. 6142101240203). (Corresponding author: Zhiqiang Pu.)

References

Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G.; McGrew, B.; and Mordatch, I. 2019. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*.

Bard, N.; Foerster, J. N.; Chandar, S.; Burch, N.; Lanctot, M.; Song, H. F.; Parisotto, E.; Dumoulin, V.; Moitra, S.; Hughes, E.; et al. 2020. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280: 103216.

Beeching, E.; Dibangoye, J.; Simonin, O.; and Wolf, C. 2021. Godot Reinforcement Learning Agents. *arXiv preprint arXiv:2112.03636*.

Booth, J.; and Booth, J. 2019. Marathon environments: Multi-agent continuous control benchmarks in a modern video game engine. *arXiv preprint arXiv:1902.09097*.

Boyd, R. 2017. *Implementing reinforcement learning in unreal engine 4 with blueprint*. Ph.D. thesis, University Honors College, Middle Tennessee State University.

Boyd, R. A.; and Barbosa, S. E. 2017. Reinforcement learning for all: An implementation using unreal engine blueprint. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 787–792. IEEE.

Chen, Y.-J.; Chang, D.-K.; and Zhang, C. 2020. Autonomous tracking using a swarm of UAVs: A constrained multi-agent reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 69(11): 13702–13717.

Ellis, B.; Moalla, S.; Samvelyan, M.; Sun, M.; Mahajan, A.; Foerster, J. N.; and Whiteson, S. 2022. SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2212.07489*.

Fu, Q.; Pu, Z.; Pan, Y.; Qiu, T.; and Yi, J. 2024. Fuzzy Feedback Multi-Agent Reinforcement Learning for Adversarial Dynamic Multi-Team Competitions. *IEEE Transactions on Fuzzy Systems*.

Gronauer, S.; and Diepold, K. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55: 895–943.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Hu, J.; Jiang, S.; Harding, S. A.; Wu, H.; and Liao, S.-w. 2021. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*.

Juliani, A. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. 2018. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, 651–673. PMLR.

Kaup, M.; Wolff, C.; Hwang, H.; Mayer, J.; and Bruni, E. 2024. A review of nine physics engines for reinforcement learning research. *arXiv preprint arXiv:2407.08590*.

Kurach, K.; Raichuk, A.; Stanczyk, P.; Zajac, M.; Bachem, O.; Espeholt, L.; Riquelme, C.; Vincent, D.; Michalski, M.; Bousquet, O.; et al. 2020. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 4501–4510.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, 157–163. Elsevier.

Liu, L.; Shao, J.; Chen, X.; Qu, Y.; Wang, B.; Ye, Z.; Tu, Y.; Qin, H.; Feng, Y. J.; Lai, L.; et al. 2023. HoK3v3: an Environment for Generalization in Heterogeneous Multi-agent Reinforcement Learning.

Lu, T.; Shu, T.; Xiao, J.; Ye, L.; Wang, J.; Peng, C.; Wei, C.; Khashabi, D.; Chellappa, R.; Yuille, A.; et al. 2024. Genex: Generating an explorable world. *arXiv preprint arXiv:2412.09624*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Mordatch, I.; and Abbeel, P. 2017. Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv preprint arXiv:1703.04908*.

NVIDIA. 2024. ACE.

Oroojlooy, A.; and Hajinezhad, D. 2023. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11): 13677–13722.

Peng, B.; Rashid, T.; Schroeder de Witt, C.; Kamienny, P.-A.; Torr, P.; Böhmer, W.; and Whiteson, S. 2021a. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34: 12208–12221.

Peng, Z.; Li, Q.; Hui, K. M.; Liu, C.; and Zhou, B. 2021b. Learning to simulate self-driven particles system with coordinated policy optimization. volume 34, 10784–10797.

Rashid, T.; Farquhar, G.; Peng, B.; and Whiteson, S. 2020a. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33: 10199–10210.

Rashid, T.; Samvelyan, M.; De Witt, C. S.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2020b. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1): 7234–7284.

Rutherford, A.; Ellis, B.; Gallici, M.; Cook, J.; Lupu, A.; Ingvarsson, G.; Willi, T.; Hammond, R.; Khan, A.; de Witt, C. S.; et al. 2024. Jaxmarl: Multi-agent rl environments and algorithms in jax. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Samvelyan, M.; Rashid, T.; de Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G. J.; Hung, C.-M.; Torr, P. H. S.; Foerster, J.; and Whiteson, S. 2019. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043.

Sapio, F.; and Ratini, R. 2022. Developing and testing a new reinforcement learning toolkit with unreal engine. In *International Conference on Human-Computer Interaction*, 317–334. Springer.

Son, K.; Kim, D.; Kang, W. J.; Hostallero, D. E.; and Yi, Y. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, 5887–5896. PMLR.

Suarez, J.; Du, Y.; Zhu, C.; Mordatch, I.; and Isola, P. 2021. The neural mmo platform for massively multiagent research. *arXiv preprint arXiv:2110.07594*.

Terry, J.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sullivan, R.; Santos, L. S.; Dieffendahl, C.; Horsch, C.; Perez-Vicente, R.; et al. 2021. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.

Wang, J.; Ren, Z.; Liu, T.; Yu, Y.; and Zhang, C. 2020. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*.

Wheeler, J. B. 2023. Reinforcement Learning Framework For The Unreal Engine.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35: 24611–24624.

Zhang, M.; Zhang, S.; Yang, Z.; Chen, L.; Zheng, J.; Yang, C.; Li, C.; Zhou, H.; Niu, Y.; and Liu, Y. 2022. Gobigger: A scalable platform for cooperative-competitive multi-agent interactive simulation. In *The Eleventh International Conference on Learning Representations*.

Zheng, L.; Yang, J.; Cai, H.; Zhou, M.; Zhang, W.; Wang, J.; and Yu, Y. 2018. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Zhong, Y.; Kuba, J. G.; Feng, X.; Hu, S.; Ji, J.; and Yang, Y. 2024. Heterogeneous-agent reinforcement learning. *Journal of Machine Learning Research*, 25(1-67): 1.