WILD-SCAV: Benchmarking FPS Gaming AI on Unity3D-based Environments

Xi Chen
Inspir.AI
chenxi@inspirai.com

Tianyu Shi
University of Toronto
ty.shi@mail.utoronto.ca

Qingpeng Zhao
Nanjing University
mf20150077@smail.nju.edu.cn

Yuchen Sun
Inspir.AI
sunyuchen@inspirai.com

Yunfei Gao
Inspir.AI
claude@inspirai.com

Xiangjun Wang
Inspir.AI
xj@inspirai.com

Abstract

Recent advances in deep reinforcement learning (RL) have demonstrated complex decision-making capabilities in simulation environments such as Arcade Learning Environment [1], MuJoCo [2] and ViZDoom [3]. However, they are hardly extensible to more complicated problems, mainly due to the lack of complexity and variations in the environments they are trained and tested on. Furthermore, they are not extensible to an open world environment to facilitate long-term exploration research. To learn realistic task-solving capabilities, we need to develop an environment with greater diversity and complexity. We developed WILD-SCAV, a powerful and extensible environment based on a 3D open-world FPS (First-Person Shooter) game to bridge the gap. It provides realistic 3D environments of variable complexity, various tasks, and multiple modes of interaction, where agents can learn to perceive 3D environments, navigate and plan, compete and cooperate in a human-like manner. WILD-SCAV also supports different complexities, such as configurable maps with different terrains, building structures and distributions, and multi-agent settings with cooperative and competitive tasks. The experimental results on configurable complexity, multi-tasking, and multi-agent scenarios demonstrate the effectiveness of WILD-SCAV in benchmarking various RL algorithms, as well as it is potential to give rise to intelligent agents with generalized task-solving abilities. The link to our open-sourced code can be found here.

1 Introduction

A challenging benchmark is essential to fully assess the capabilities of deep reinforcement learning (RL) algorithms. Many simulators have been used to challenge and evaluate RL algorithms, bringing significant contributions and development to the RL community. For example, the Arcade Learning Environment (ALE) [1] presents a valuable test-bed with a collection of Atari 2600 games to evaluate RL algorithms. Deep Q Network (DQN) successfully learns to play the Atari 2600 games given screen pixels as inputs and achieves outstanding performance on ALE [4]. Other examples, e.g., Dueling DQN [5], Prioritized DQN [6], Quantile Regression DQN [7], also largely benefited from ALE as

[https://github.com/inspirai/wilderness-scavenger](https://github.com/inspirai/wilderness-scavenger)

Preprint. Under review.
they were developed. To support more general task scenarios, Gym [8], built by OpenAI, exposes standard interfaces for testing different RL algorithms over a variety of environments, allowing researchers to easily compare the performance of alternative approaches.

Recently, RL methods have demonstrated superior ability to humans on multiple Atari tasks. DeepMind built Agent57 [9], which is the first deep reinforcement learning model that surpasses human baseline performance on all Atari games. The success of Agent57 comes from its meta-controller design to balance efficient exploration and exploitation. As we are building more and more intelligent task-solving agents, it appears that memory utilization [10], curiosity seeking [11], and exploration-exploitation balancing [9] have become some of the key points to motivate these developments.

To explore the potential of RL algorithms and push forward the performance limits, the community keeps seeking more challenging environments that generally involve real-world tasks and corresponding interactions. For example, navigation and resource collection tasks [12] are very common in the real world and are integrated into several environments, e.g., Coin-Run [13], Deepmind Lab [14]. Furthermore, by introducing more comprehensive forms of observational inputs and more flexible action types, agents can be evaluated on not only navigation and resource collection but also more complex tasks like the move-and-shoot task in ViZDoom [3]. While adding action options may lead to an exponential increase in the complexity of the policy, several advanced RL algorithms have been proposed to solve these challenges. For instance, hierarchical Reinforcement Learning (HRL) [15] has been proposed to solve the huge action space and exploration challenges, in which the high-level layer learns policy through options while the low-level layer executes basic options, such as motion, attack, tool, and resources.

Following the success of ViZDoom, training intelligent agents with general task-solving capabilities in open-world environments has attracted increasing attention. However, the lack of adequate training and testing benchmark environments remains an obstacle to research in this field. ViZDoom is a useful and fast 3D environment but still oversimplified in both the visual style and the world composition compared to the simulation appearance in modern 3D games, which is relatively detached from the more practical side of the community’s needs. In this context, the goal of WILD-SCAV is to advance research in the field of open-world intelligent agent learning. As a stepping stone to the ultimate goal of learning highly intelligent agents’ “living” in the virtual world, we decided to first focus on open-world FPS games. With the popularity of battle royale games, such an environment can serve as a flexible and open playground for both algorithmic research in RL and solution optimization in the game industry.

To the best of our knowledge, WILD-SCAV is the first FPS-based environment that allows agents to explore in such a 3D world environment. Our main contributions are listed as follows:

- WILD-SCAV provides a customizable 3D environment, where the ground landscape, the structure of houses, and the placement of various types of objects can be generated with PCG (Procedural Content Generation) technique [16], which is very suitable for NPC research [12].
- WILD-SCAV focuses on learning intelligent agents in the open-world environment. It provides more spaces for exploration, more flexibility in task design and open challenges for generalization with diverse learning scenarios.
- WILD-SCAV supports agent training in multi-task and multi-agent scenarios, similar to the common setup in recently popular battle royale games (e.g., PUBG [17]), which includes tasks like random target navigation, competitive/cooperative resources gathering, and free fight etc.

2 Background: 3D open-world FPS game environment

AI-powered next-generation gaming experiences for open-world games have attracted increasing attention after the success of AI in StarCraft II [18] and DOTA [19] as the next big challenge. However, the lack of satisfactory testing environments remains an obstacle to research in this area. In this context, the goal of this competition is to promote research on intelligent agent learning for 3D open-world FPS games. In 3D open-world environments, an agent perceives its environment in a human-like manner, using visual scenes as input. Intelligent agents are expected to integrate visual perception and contextual game features, process incomplete information, deal with the dynamic
variation of environments and multiplayer enemies, and then perform long-term planning. In addition, to maximize scores on the tasks set, agents must generalize their learned skills to unknown test environments. For the competition, we provide an FPS game environment similar to popular Battle Royale games (e.g., PUBG), where multiple players compete against each other for limited supply resources. We will evaluate each trained agent on the randomly generated battlegrounds.

3 WILD-SCAV Environment

3.1 Observation Space

The gameplay interface provides multiple sources of information about the agent (e.g., location, orientation) and its surrounding environment. The observation mainly consists of two parts, visual perception inputs and game variables.

**Visual perception.** We implemented an efficient way to compute a low-resolution depth map (e.g. $10 \times 10$) from the agent’s camera using only the location, orientation values, and mesh data of the static scenes, to enable efficient feature learning in the scenario of large 3D open worlds. In this way, we trade-off between granularity of visual information and computation efficiency to allow researchers to get most out of this new environment when using it for agent training. However, to make our environment more usable for more generalized tasks, we also added support for calculating high-resolution depth maps, LIDAR detection, and the first person camera images, as shown in Figure 1 and Figure 2. A short video visualization of the camera images can be found here.

**Game variables.** We also provide access to multiple game variable information. The variables include current location and orientation of the agent, state of motion (e.g. on ground or in the air), health, state of combat (e.g. firing, being hit), and task-related metrics (e.g. target location, number of collected resources). These game variables can be essential to task-solving in more complex tasks.

Figure 1: **left:** depth map visualization example. **right:** schematic diagram of LIDAR perception. The arrows means the depth map is a 360-degree panorama.

Figure 2: Images of the agent’s first person camera from different viewing angles.

The above observations provide rich and efficient information for the agent to learn optimal policies. Users have the freedom to construct customized observational features for different training tasks.

---

[https://youtu.be/VrXPFOHoIbU](https://youtu.be/VrXPFOHoIbU)
3.2 Action Space

WILD-SCAV provides comprehensive and structured action spaces. Users can easily explore the action spaces to facilitate skill learning of the agent. To be more specific, it support actions for navigation task, such as walking direction (WALK_DIR); walking speed (WALK_SPEED). It also supports camera angle direction change for wider range of observation, such as horizontal camera angle (yaw) between two frames (TURN_LR_DELTA) and vertical camera angle (pitch) between two frames (LOOK_UD_DELTA).

Unlike previous shooting game environments [3], our environment has more complex terrain, where agents have huge space of movement trajectories when moving around. An agent can jump up stairs (JUMP) and pick up randomly spawned resources (PICKUP). These characteristics of WILD-SCAV make it a favorable testbed for cooperative and competitive resource gathering tasks.

In addition, in contrast to multi-agent cooperative and competitive environments like Multi-Agent Particle environment [20], we also introduce the combat game system to improve the complexity of the environment. There are two combat actions: (a) whether to fire the weapon and cost one bullet at the current time step (SHOOT) and (b) whether to refill the weapon’s clip using spare ammo (RELOAD). The detailed tasks illustration will be analyzed in 3.3 and a summarized action space description for each task is shown in Table 3.2.

| Action Class       | Navigation | Supply Gathering | Supply Battle | Type     | Range           |
|--------------------|------------|------------------|---------------|----------|-----------------|
| WALK_DIR           | ✓          | ✓                | ✓             | float    | [0., 360.]      |
| WALK_SPEED         | ✓          | ✓                | ✓             | integer  | [0, 10]         |
| TURN_LR_DELTA      | ✓          | ✓                | ✓             | float    | \([-∞, ∞]\)     |
| LOOK_UD_DELTA      | ✓          | ✓                | ✓             | float    | \([-∞, ∞]\)     |
| JUMP               | ✓          | ✓                | ✓             | boolean  | True / False    |
| PICKUP             | x          | ✓                | ✓             | boolean  | True / False    |
| SHOOT              | x          | x                | ✓             | boolean  | True / False    |
| RELOAD             | x          | x                | ✓             | boolean  | True / False    |

Table 1: Action space description for the typical tasks

3.3 Typical Tasks

In this section, we will discuss some typical tasks supported by the WILD-SCAV environment. The basic tasks are navigation, supply gathering, and battle. By combinations of different typical tasks, different experiment scenarios can be created to validate the performance of different RL algorithms.

3.3.1 Navigation

The challenge for the agent is to navigate as quickly as possible from a starting location to a destination (both randomly selected) in a randomly generated open world. The world consists of various structures, such as buildings, trees, rocks, and lakes. Ideally, an efficient and general navigation strategy must be able to skillfully use these elements and flexibly adapt to new worlds that have not been seen yet.

- **State:** The observation is the target position, current position and the depth image.
- **Action:** Walking direction, walking speed, camera angle (yaw, pitch), and jump.
- **Reward:** The agent is rewarded when it reaches the target point.
- **Evaluation criteria:** Time consumed to reach the target, i.e., the episode length.

3.3.2 Supply Gathering

The challenge for the agent is to collect as many supplies as possible by opening the blue supply boxes in a randomly generated open world with unknown supply distribution. Supply boxes may appear at any accessible location in the open world, for example, on outdoor grounds or on a certain
floor of a house, hiding behind a tree or a stone. To make it even more challenging, we have designed a special mechanism to determine the number of supplies stored in each supply box. In general, a supply box inside a building contains a significantly higher number of supplies than an outdoor box, while the number of supply boxes inside buildings could be relatively small. These agents are encouraged to explore both outdoor areas and indoor spaces to maximize their collection of supplies. Also, the actual supply quantity in each box has some randomness but is constrained within given ranges.

- **State:** The observation are the locations of all nearby supplies, the specific number of supplies at each location, the agent’s current location and the depth image.
- **Action:** Walking direction, walking speed, camera angle (yaw, pitch), and jump.
- **Reward:** To add variability to the task, we can set different termination criteria. We design two separate sub tasks. In the first task, the agent will get a high reward if it can collect supply as many as possible. In the second task, the agent is rewarded for collecting the target supply as fast as possible.

- **Evaluation criteria:** In the first sub-task, we use the number of collected supplies to evaluate. In the second sub-task, we use episode length as evaluation criteria.

### 3.3.3 Supply Battle

Based on the supply gathering, multiple agents are dropped into a large, randomly generated world (terrain, buildings, plants, supplies) and compete for supplies. These agents can fight with weapons and respawn when killed. During combat, agents are encouraged to take cover to ensure their safety. When an agent dies, some of his supplies drop and can be collected by other agents. Since the total number of supplies in the game is limited, agents should learn to optimize their search strategy to collect supplies faster. With the introduction of the combat system, agents can also use their combat advantage to snatch supplies from other agents to speed up the accumulation of supplies.

The observed state and actions of the agent are very similar to the material collection task of the previous task or are added based on it. In addition to its own attributes in the gathering task, observation can also include the state of other agents in the field of view, but if the agent is not in the field of view, this state cannot be obtained, which leads to the non-stationarity and partial observation of the environment, making it more complex and difficult to learn. In addition to basic actions such as movement, we further extend the action types by adding the behavior of shooting attacks and changing bullets. It is expected to learn a strategy that can obtain their supplies by killing other agents and also avoid being killed by other agents.

### 4 Experiments

In this section, we present the results and analysis of the experiments based on the combination of the elements of the typical task from the above section 3.3. Note that we could extend more combinations of different typical tasks, but for this paper, we decided to focus on several representative experiments and demonstrate how our environment can be configured to assess the performance of different RL methods across different learning scenarios.

For a fair comparison, we adopt the same resource configuration and fixed hyper-parameters for all tasks and experiments to test the adaptability and scalability of the compared algorithms. We use the same machine with 96 Intel(R) 8163 (2.50GHz) CPU cores to run all algorithms. For simplicity and better reproducibility, we construct all training pipelines using the standard trainer APIs from the commonly used RLlib [21] among the research community.

To keep it simple and clear, for the benchmark experiments in this paper, we only select three commonly used RL algorithms, including IMPALA [22] Proximal Policy Optimization (PPO) [23] and Asynchronous Advantage Actor-Critic (A3C) [24]. For each episode rollout during training, while it is possible to let the agent explore the environment for a long time, we can definitely speed up the learning by setting appropriate episode lengths $T$ for different tasks. In our experiments, we empirically choose the episode length for each task. Specifically, we fix the maximum episode length to $T = 400$ for the single-agent navigation and the multi-agent target capture task, and we fix the episode length to $T = 600$ for the multi-agent supply gathering task.

#### 4.1 Single-Agent Navigation

For an open-world 3D environment, the navigation task is the basis for completing other more complex tasks. For navigation tasks, even if the environment is very complex, the agent can quickly learn a relatively good strategy to reach any target point thanks to the easy randomization of the start location. The agent is given a reward of +1 if it reaches the target, and 0 otherwise. However, such a reward distribution can be very sparse, especially for large map sizes.

Figure 6 shows the task training curve of the evaluated algorithms for different map sizes and scene complexity. Overall, for tasks of different complexity and different sizes, all algorithms can
quickly find some decent strategies to reach the target point, and they will always slowly explore new strategies to reach the target in faster ways.

Seen from the results in Figure 6 and Table 2, with large size and higher scene complexity of the map, the average episode reward becomes smaller, resulting in more “slow-learning” steps. To help improve the learning efficiency, one promising approach is to design better Intrinsic curiosity-driven reward to encourage more efficient exploration [25].

In addition, both A3C and PPO can learn a good strategy, while the performance of IMPALA is relatively poor. PPO performs best in maps with a small size and shows the highest improvement speed at the early stage, while A3C performs better under larger sizes of the map and may need longer time steps to reach a high-performance level. IMPALA in this task indeed shows steady growth of performance but has a much slower speed and higher fluctuation.

| Map   | A3C Mean (SD) | A3C SuccRate | PPO Mean (SD) | PPO SuccRate | IMPALA Mean (SD) | IMPALA SuccRate |
|-------|---------------|---------------|---------------|--------------|-----------------|-----------------|
| 101   | 219.4 (132.8) | 0.73          | 228.4 (140.1) | 0.69         | 330.9 (124.4)   | 0.27            |
| 102   | 185.2 (111.5) | 0.90          | 170.6 (115.0) | 0.90         | 330.5 (119.8)   | 0.29            |
| 103   | 158.2 (85.4)  | 0.96          | 87.5 (73.6)   | 0.97         | 136.8 (130.8)   | 0.82            |
| 104   | 118.8 (77.9)  | 0.96          | 63.9 (45.4)   | 0.99         | 109.5 (122.5)   | 0.86            |
| 008   | 255.9 (126.9) | 0.71          | 283.1 (118.6) | 0.66         | 354.1 (105.0)   | 0.19            |
| 014   | 239.6 (124.3) | 0.74          | 284.0 (121.7) | 0.62         | 353.4 (97.8)    | 0.23            |

Table 2: Evaluation results on the single agent navigation task. We compare average episode lengths and success rates achieved with the learned policies on different maps. For the episode length, we report both the mean value and the standard deviation (in the brackets).

4.2 Cooperative Supply Gathering.

The second task is a multiplayer cooperative task. In this task, multiple agents are taking action as a team to collect as many supplies as possible within an episode. In this task, the agent is trying to learn a policy that jointly considers their own states (e.g., current agent location), the supply state (e.g., location of a nearby supply), and the teammate state (e.g., location of other agents) to collect the
supplies in a more efficient way than the single-agent scenario. We run all the experiments on map 101, where randomly generated supplies are distributed denser near the center of the map and sparser far away from the center. For each supply collected, the agent is given a small positive reward.

Figure 7 shows the change in the average number of supplies collected in an episode during training. We can see that PPO learns to handle the task quite fast, with only $2 \times 10^6$ time steps when reaching an acceptable level of performance. IMPALA has the lowest improvement speed at the start of training but shows a relatively steady and moderate speed of performance growth. However, A3C performs not very well and seems to require more time steps of experience to basically handle the task. The results may have shown the low exploration efficiency of A3C in case of sparse reward distribution and that PPO can be a favorable choice in such an open-world environment.

In such a multi-agent case, how to coordinate the actions of all agents in order to collect the most supplies in a limited time is still a challenging problem. Although here we only test the performance of three commonly used algorithms, some recently proposed MARL algorithms [26, 27] can be simply applied with little modification of the environment. From the results in Table 3, we can see that the best method, PPO, can only reach a level of around 100 average supplies while there are more than 200 supplies on the entire map. This may be due to the exploration difficulty that agents are more likely to collect supplies in a small sub-region where supplies are relatively close to the agents’ current locations rather than explore large open areas. But some “treasure” places with more densely distributed supplies can be far away from the start locations. As the number of agents increases, supplies in a sub-region can be quickly depleted by a small group of agents, leaving other agents a sparser reward distribution, which suggests that the environmental setups should be more subtle on the multi-agent task when the number of agents increase, to avoid insufficient local optimal.

| A3C | PPO | IMPALA |
|-----|-----|--------|
| Map | SupplyNum | SupplyNum | SupplyNum |
| 101 | 49.49 (18.67) | 98.92 (52.62) | 91.68 (45.90) |

Table 3: Evaluation results on the cooperation supply gathering task. We report both the mean and standard deviation (in the brackets) of the total collected supply numbers in 3 minutes with 4 agents.

4.3 Cooperative Target Capture

The last task is also a multiplayer cooperative task, where multiple agents are trying to capture the target supply as fast as possible cooperatively. The supply is hidden at some point in the open world. Once an agent captures it, the episode is considered a success and finished, and a +1 reward is given to this agent.

In Figure 8, we show the learning results of different methods where PPO can achieve the highest reward. When collecting the state and supply position of each agent, the strategy will assign the nearest agent to quickly catch this supply. For the same map configuration, such a multi-agent environment can solve the task faster than a single-agent environment compared to the results in Table 4. Similar to previous experiments, we report the mean and standard deviation in the bracket. Note that the standard deviation is a bit high, which is because we randomly generate the agent in the map without fixed agents’ start and target locations. To conclude, we find that PPO once again achieves the best performance with the smallest episode length.

| A3C | PPO | IMPALA |
|-----|-----|--------|
| Map | EpsLen | EpsLen | EpsLen |
| 101 | 143.18 (122.71) | 94.69 (98.55) | 251.58 (166.72) |

Table 4: Evaluation results on the multi-agent target capture task. We report both the mean value and standard deviation of the episode length to reach the target location.
5 Related work

In general, modern 3D FPS games are inherently incomplete information games that are extremely hard to learn winning strategies in multiple-player scenarios and are known to have no optimal policy. Despite the difficulties, there have been attempts over the past decade to apply reinforcement learning in FPS games. To our best knowledge, the most influential work is the Augmented DRQN model [28], where the method leverages both visual input and the game feature information (e.g., presence of enemies or items) and modularizes the model architecture to incorporate independent networks to handle different phases of the game. Their approach successfully learned a competitive FPS agent by minimizing a Q-learning objective and showed better performance than average human players. Following this success, more work on learning FPS game agents has been proposed, such as Arnold [29] which benefits from the Action-Navigation architecture, Divide and Conquer deep reinforcement learning [30] which further refined the idea of separating the control strategies of map exploration from enemy combat. Although these methods have shown promising results, their training and evaluation context is largely limited to old-fashioned video games with relatively small world sizes, such as VizDoom (originally 1993) and Quake 3 (originally 1999). Recently, Pearce and Zhu [31] tried to learn an FPS agent to play CSGO, a phenomenal modern 3D FPS game with high-resolution visual rendering. This new game environment not only introduces more computational burden (mostly due to extracting visual features) but also makes it more difficult for the agent to explore and adapt to the game world efficiently. The new approach addressed the challenge primarily by using behavioral cloning, and the learned agents showed reasonably good performance compared to normal human players in the Deathmatch mode. Note that there are also other 3D simulators, such as Mujoco [2], DeepMind Lab [14], etc. They are not extensible to more complicated real-world problems. On the other hand, our simulator is more suitable for open-world exploration. Recently, MINEDOJO has been developed with thousands of diverse open-ended tasks [32]. With MINEDOJO’s data, one can leverage large pre-trained video language models to learn reward functions and then guide agent learning in various tasks. Building upon these works, we seek to further expand the frontiers of intelligent agent learning in modern large-scale open-world games.

6 Future work

As discussed in Section 3.3 we can create great combinations of different tasks (i.e., navigation, supply gathering, battle) to support various experiments. Besides these representative experiments, as discussed in Section 4, we also tried other experiments. However, we find that traditional methods,
as discussed in Section 4 will fail in more challenging experiments. For example, we randomly generated ten agents on map 103, as shown in Figure 9. These agents are controlled by the random policy while we train another agent, which is controlled by A3C. However, we find that the trained agent is hard to learn an appropriate behavior because it cannot go to the target point successfully.

We suggest that there are a few potential research ideas for future improvement. Firstly, when the map is becoming larger, the agent will become hard to find an enemy, which will be a barrier to efficient exploration. Curiosity-driven reinforcement learning method can be utilized to encourage exploration in sparse reward setting [33, 34]. Secondly, we find that when the agents’ number increase, the learned agent is hard to process in such a large observation space, and the learning becomes very difficult. In the future, we can investigate how to better capture the mutual interplay of different agents through communication, such as graph neural network [35]. Lastly, as WILD-SCAV can support Procedural Level Generation (PCG), it is also worth studying PCG-based reinforcement learning to improve the generalizability of our agent [36].

7 Conclusion

In this paper, to bridge the gap with realistic NPC gaming problems, we present WILD-SCAV, the first realistic 3D FPS-based environment, with the support for configurable complexities, multi-tasks, and multi-agents. Built upon PCG world generation techniques, WILD-SCAV enables the configurations of object types, areas, locations, orientations, etc. We have created several demo tasks, i.e., navigation, supply gathering, and supply battle, to evaluate the performance of different RL methods and enable researchers to develop more powerful algorithms through configurable environments. To facilitate further research development, we open-sourced our code for the simulator and training agents with A3C, PPO, and IMPALA. We also host the open-world FPS game AI competition to attract global researchers to innovate on the algorithms. Winners will be selected based on the evaluation of randomly generated environments. We believe WILD-SCAV could further push forward the development of AI algorithms in the 3D Open World, bringing it a step closer to intelligent and generalized task-solving agents in realistic NPC research.

References

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.

[2] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems, pages 5026–5033. IEEE, 2012.

[3] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In 2016 IEEE conference on computational intelligence and games (CIG), pages 1–8. IEEE, 2016.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In International conference on machine learning, pages 1995–2003. PMLR, 2016.

[6] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[7] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.
[9] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In International Conference on Machine Learning, pages 507–517. PMLR, 2020.

[10] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In International conference on learning representations, 2018.

[11] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. arXiv preprint arXiv:2002.06038, 2020.

[12] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. Deep reinforcement learning for navigation in aaa video games. arXiv preprint arXiv:2011.04764, 2020.

[13] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In International Conference on Machine Learning, pages 1282–1289. PMLR, 2019.

[14] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdès, Amir Sadik, et al. Deepmind lab. arXiv preprint arXiv:1612.03801, 2016.

[15] Wikipedia. PUBG: Battlegrounds — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=PUBG%3A%20Battlegrounds&oldid=1090843995, 2022. [Online; accessed 09-June-2022].

[16] Julian Togelius, Alex J Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O Stanley. Procedural content generation: Goals, challenges and actionable steps. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[17] Wikipedia. PUBG: Battlegrounds — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=PUBG%3A%20Battlegrounds&oldid=1090843995, 2022. [Online; accessed 09-June-2022].

[18] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782, 2017.

[19] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3974–3983, 2018.

[20] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in neural information processing systems, 30, 2017.

[21] Eric Liang, Richard Liaw, Robert Nishihiara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In International Conference on Machine Learning, pages 3053–3062. PMLR, 2018.

[22] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In International Conference on Machine Learning, pages 1407–1416. PMLR, 2018.

[23] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. arXiv preprint arXiv:1611.06256, 2016.
[25] Joshua Hare. Dealing with sparse rewards in reinforcement learning. *arXiv preprint arXiv:1910.09281*, 2019.

[26] Weichao Mao, Lin Yang, Kaiqing Zhang, and Tamer Basar. On improving model-free algorithms for decentralized multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 15007–15049. PMLR, 2022.

[27] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939*, 2020.

[28] Guillaume Lampe and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[29] Devendra Singh Chaplot and Guillaume Lampe. Arnold: An autonomous agent to play fps games. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[30] Georgios Papoudakis, Kyriacos C Chatzidimitriou, and Pericles A Mitkas. Deep reinforcement learning for doom using unsupervised auxiliary tasks. *arXiv preprint arXiv:1807.01960*, 2018.

[31] Tim Pearce and Jun Zhu. Counter-strike deathmatch with large-scale behavioural cloning. *arXiv preprint arXiv:2104.04258*, 2021.

[32] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*, 2022.

[33] Susanne Still and Doina Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148, 2012.

[34] Lulu Zheng, Jiarui Chen, Jianhao Wang, Jiamin He, Yujing Hu, Yingfeng Chen, Changjie Fan, Yang Gao, and Chongjie Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *Advances in Neural Information Processing Systems*, 34:3757–3769, 2021.

[35] Tianyu Shi, Jiawei Wang, Yuankai Wu, Luis Miranda-Moreno, and Lijun Sun. Efficient connected and automated driving system with multi-agent graph reinforcement learning. *arXiv preprint arXiv:2007.02794*, 2020.

[36] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

[37] Jie Hu, Wang gen Wan, and Xiaoqing Yu. A pathfinding algorithm in real-time strategy game based on unity3d. In *2012 International Conference on Audio, Language and Image Processing*, pages 1159–1162. IEEE, 2012.

A Appendix

A.1 Generating benchmark maps with PCG

To demonstrate the potential in using our benchmark environment to learn robust agent policies with reasonably good capacity of generalization, we created the training maps with different sizes, house numbers, house densities and storey numbers. For each size of the map, we first select the number of houses, the house density and the range of the number of floors of a house. Then we randomly sample from all the legal locations where to generate the houses based on the selected density. After this, specific building structure of the house and the storey numbers are randomly selected from the predetermined asset pool. In this way, we can easily generate thousands of different maps. But in this paper we focus on a small scale evaluation where we only generate 6 maps of 3 sizes to compare the performance of different algorithms on various tasks. Figure 9 shows the top view of the generated maps and the detailed generation configurations of them are given in Table A.1.
Figure 9: Different maps visualization. We can see different terrain shapes and different housing density. For map 008 and map 014, we select two big maps. For map 101,102,103,104, we randomly generated a big map then cut out a small area of the big map.

Table 5: Generation configurations of the benchmark maps

| Map ID | Size    | House numbers | House density | Storey numbers |
|--------|---------|---------------|---------------|---------------|
| 008    | 500 × 500 | 12            | low           | 2-3           |
| 014    | 500 × 500 | 15            | high          | 2-3           |
| 101    | 200 × 200 | 4             | low           | 1             |
| 102    | 200 × 200 | 8             | high          | 2-3           |
| 103    | 100 × 100 | 2             | low           | 1             |
| 104    | 100 × 100 | 4             | high          | 2-3           |
A.2 Computation cost and simulation speed

We also evaluate the simulation speed of different numbers of workers and agents, this result is shown in table A.2. All the results are tested on a single AMD 5800H CPU core.

Table 6: Throughputs (total FPS) of the environment when running on a single CPU machine with different numbers of concurrent process and agents. The maximum throughput for each number of agents is bolded.

| # agents | # process | throughput |
|----------|-----------|-----------|
| 1        | 1         | 499.61    |
| 2        | 2         | 996.90    |
|          | 4         | 1974.84   |
|          | 6         | 2766.21   |
|          | 8         | 3506.31   |
|          | 10        | **4187.03** |
| 5        | 1         | 500.18    |
|          | 2         | 996.94    |
|          | 3         | 1609.81   |
|          | 4         | 1463.60   |
|          | 5         | 1864.10   |
|          | 6         | **2023.76** |
| 10       | 1         | 500.54    |
|          | 2         | 997.62    |
|          | 3         | 1095.73   |
|          | 4         | 960.10    |
|          | 5         | **1426.10** |
|          | 6         | 1246.24   |

A.3 Reward Configuration

We designed the reward functions for the three evaluation tasks simply by translating the agent state into a binary reward of whether reaching the goal. The detailed operations to compute the reward are given below in a form of Python code snippets.

- Single-Agent Navigation

```python
state = self.update_state()
curr_loc = self.get_location(state)
targ_loc = self.target_location
if self.get_distance(curr_loc, targ_loc) <= 1:
    reward = 1
else:
    reward = 0
```

- Cooperative task 1: supply gathering

```python
state = self.get_states(agent_id)
if state.num_supply > self.collected_supply[agent_id]:
    rewards[agent_id] = 1
    self.collected_supply[agent_id] = state.num_supply
else:
    rewards[agent_id] = 0
```

- Cooperative task 2: target capture

```python
state = self.get_states(agent_id)
curr_loc = self.get_location(state)
targ_loc = self.target_location
if self.get_distance(curr_loc, targ_loc) <= 1:
    rewards[agent_id] = 1
else:
    rewards[agent_id] = 0
```

A.4 Environment interface

WILD-SCAV can support popular Gym interface [8]. The typical interface is shown below. To facilitate the design of new algorithm and benchmark comparison, our gaming interface can support different maps, different locations, different reward design for different tasks.

```bash
% Define game backend
game = Game(
    map_dir="/path/to/map_data",
    engine_dir="/path/to/engine_backend",
)
```
% Create environment
env = gym.make(
    id="WildScav-Navigation-v0",
    env_config={
        "engine": game
        "map_id": 1,
        "timeout": 30,
        "start_location": [0, 1, 0],
        "target_location": [5, 0, 3],
        obs=[depth_image, position],
        reward= RewardManager.add('Navigation')
    }
)

A.5 Model description

We also provide the training details of the RL algorithms on WILD-SCAV using standard RLlib library [21]. We run all experiments on a Linux server with 96 CPU cores. We fix the number of experience sampler workers to 80 (1 CPU per worker) and allocate the rest of the CPUs to the master learner. We use Adam as the default optimizer for all models trained in the experiments. The detailed training configurations for experiments on PPO, A3C, IMPALA can be found in Table 7, Table A.5, Table A.5 respectively.

| Parameter Name       | Value          |
|----------------------|----------------|
| learning rate        | 0.00001        |
| train batch size     | 32000          |
| SGD mini-batch size  | 256            |
| rollout fragment length | 400     |
| entropy coefficient  | 0.01           |
| value loss coefficient | 0.5       |
| KL-divergence coefficient | 0.2    |

Table 7: Training configuration of PPO experiments

| Parameter Name       | Value          |
|----------------------|----------------|
| learning rate        | 0.0001         |
| rollout fragment length | 400     |
| entropy coefficient  | 0.01           |
| value loss coefficient | 0.5       |

Table 8: Training configuration of A3C

| Parameter Name       | Value          |
|----------------------|----------------|
| learning rate        | 0.0005         |
| train batch size     | 32000          |
| rollout fragment length | 400     |
| entropy coefficient  | 0.01           |
| value loss coefficient | 0.5       |

Table 9: Training configuration of IMPALA

A.6 Network architecture description

We use the following network architecture configuration to encode the information.

```python
# FullyConnectedNetwork
# Number of hidden layers to be used.
"fnet_hiddens": [256, 256],
# Activation function descriptor.
# Supported values are: "tanh", "relu", "swish" (or "silu"),
# "linear" (or None).
"fnet_activation": "tanh",
# VisionNetwork
```
A.7 Running time

| Map | A3C Running time (s) | PPO Running time (s) | IMPALA Running time (s) |
|-----|----------------------|----------------------|------------------------|
| 101 | 10456.62             | 29414.05             | 4470.43                |
| 102 | 10760.02             | 28100.36             | 4255.84                |
| 103 | 16922.06             | 31898.47             | 6455.39                |
| 104 | 1962.49              | 9855.69              | 7014.64                |
| 008 | 6076.44              | 28854.93             | 4211.52                |
| 014 | 6352.47              | 28305.63             | 4278.53                |

Table 10: Evaluation results on the navigation task. We report the total running time in second.

| Map | A3C Running time (s) | PPO Running time (s) | IMPALA Running time (s) |
|-----|----------------------|----------------------|------------------------|
| 101 | 1925.99              | 12250.53             | 2312.53                |

Table 11: Evaluation results on the cooperative supply gathering. We report the total running time in second.

| Map | A3C Running time (s) | PPO Running time (s) | IMPALA Running time (s) |
|-----|----------------------|----------------------|------------------------|
| 101 | 1300.56              | 16885.55             | 1748.60                |

Table 12: Evaluation results on the cooperative target capture. We report the total running time in second.

B Winning Policy Analysis

We held an online competition in the past few months to test the RL algorithm for solving this problem. Based on our previous results in the competition[3], we analyze the winning policy in the leader board below. Compared to the rule-based method, such as A star algorithm[37], we can find that the traditional reinforcement learning agent cannot efficiently handle this task. Therefore, we analyze the winning strategies from the top submissions.

B.1 Navigation

The goal of this task is to make the agent reach the target as soon as possible. The challenge is that in this track, the spawn point and destination location basically do not involve indoors, but the agent still has the possibility of entering the room by mistake. Some destinations are set on hillsides, and the agent needs to know how to climb hills.

After testing the map, we found that the extreme value of the boundary coordinates does not exceed ±300, and the tallest building does not exceed four floors. Therefore, we established a 600*600*4

[https://sites.google.com/view/inspirai-wildscav-cog2022/home]
obstacle marker matrix for sampling and recording terrain information. There are several tricks to solve these tasks:

- During the movement of the agent, the current position will be stored as a checkpoint every time given a certain number of time steps. If the agent fails to move a valid distance for a long time, it will try to return to the previous saved checkpoint.
- When we use visual depth image to detect an obstacle, the slope of the obstacle is also calculated. We assume there is no obstacle if the slope is low.
- When the agent is walking, if there is obstacle but there is no obstacle in the image, we will try to jump.

B.2 Supply gathering

In the second task, the agent needs to collect as many supplies as possible on the map in a fixed time. There are a large number of indoor scenes on the map, which requires the agent to have the ability to explore complex indoor scenes. The buildings on the map have different floors, and the spawn point may also be located on the upper floor. There are several tricks to solve this task:

- The supply gathering problem can be decomposed into multiple navigation sub-problems. In each step, we maintain a list of information, such as supply points, and location information, to make the decision.
- The agent will use a decision tree policy to determine when and where to collect supply.
- In the high-floor area, the agent will record the exploration route and return to the original route first when it is necessary to go downstairs.

B.3 Supply battle

In the third task, compared with supply gathering, multiple agents are competing on the same map, and it is necessary to consider killing other agents to gain more resources. There are a large number of indoor scenes on the map, which requires the agent to have the ability to explore complex indoor scenes. The buildings on the map have different floors, and the spawn point may also be located on the upper floor. Agents need to have certain attacking capabilities, such as shooting other agents.

- If the agent detects local information, it will automatically aim at the nearest enemy and fire.
- We will consider reloading when the current clip is lower than a threshold.

C Further information about our gaming engine

Our environment consists of two major components, the game engine (backend) and gameplay API (frontend). We have described the gameplay interface and training algorithm configurations in previous sections. In this section, we provide more information on how we build the infrastructure and optimize the simulation throughput.

Communication. we use GRPC protocol to design the format and content of observation, action, environmental control command and event messages. The protocol will convert data from both ends into messages with a uniform format so that they can be transmitted between processes across multiple languages. To ensure synchronous control of the game simulation, we built a simple message queue server on the frontend side to receive the observation message from the engine, wait for the algorithm level to consume, and then send back the new action to the engine.

Simulation optimization. We consider the following strategies to optimize the gaming performance:

- The game scene is too large, and there are too many resources, resulting in excessive rendering pressure and causing stuttering. As a result, we reduce the rendering batch of the game from more than 3000 to an average of around 200.

[https://grpc.io/]
• On the premise of meeting the requirements of the algorithm, the communication request of GRPC is reduced. Therefore, the running speed of the engine is improved, and the training rate is accelerated.

• We optimize the management of the protocol data to reduce running consumption caused by game memory management.