Contents

1 Dr. Yielding's Questions

While HARL is indeed a subfield of MARL, the distinction between the two can often be unclear or inconsistently used in the literature. In the broadest sense, 'heterogeneous' may refer to any MARL scenario where the agents are not identical. However, using this broad definition risks diluting the practical significance of the term.

To provide a more useful distinction, HARL should be invoked when the heterogeneity of the agents is fundamental, essential, or definitional rather than merely incidental. This means the differences among agents are critical to their roles and interactions within the system, not just minor variations.

Prior to my literature review, I would have considered HARL to apply specifically to cases where agents are distinct from the outset, either in their capabilities (For action-space $\mathcal{A}$, $\mathcal{A}_1 \neq \mathcal{A}_2$) or their observation spaces ($\mathcal{O}_1 \neq \mathcal{O}_2$). This intrinsic heterogeneity is evident in works like [1] (the Irish conference paper mentioned in Dr. Yielding's question) and implicitly reflected in [2] (OpenAI Five), even though they do not explicitly label their methods as HARL.

The most comprehensive source on HARL usage is Zhong et al. [3], whose work has greatly influenced my understanding. They focus on implementing algorithms that encourage the development of heterogeneous policies among agents. Their framework increases the likelihood of individual agents converging on distinct policies, which I term emergent heterogeneity.

In my prospectus, I also mention a suspicion that this approach might not entirely prevent agents from converging on policies that are functionally similar, thus lacking true diversity. However, this assertion remains an ancillary detail as it is not yet substantiated by empirical evidence.

Therefore, I propose distinguishing between intrinsic heterogeneity, where agents are fundamentally different before training, and emergent heterogeneity, where differences arise as a result of the learning process. In the cases where the heterogeneity of the agents falls below a level of functional relevance, and appears to be incidental, I argue that they should not be labeled as HARL. By maintaining these distinctions, we can more accurately categorize and understand the applications and implications of HARL and MARL.

Below, I apply these distinctions to the cases proposed:

1.1 HAA2C and HADDPG are among the numerous algorithms proposed by Zhong et al. [3]. While it seems reasonable that their algorithms could be applied to situations with intrinsic heterogeneity, their implementations and tests are applied to environments with agents that are functionally the same.

I intend to implement (at least a subset of) their algorithms to the extent that time allows. In doing so, I hope to either corroborate or contradict their results by comparing them to similar algorithms under different conditions. This exploration aims to identify any apparent advantages of these algorithms when applied to the experimental variables proposed for Contribution 1. These experimental variables represent smaller difficulties that we expect to face in Contributions 2 and 3.

1.2 (a) The scenario described in this part of the question is, perhaps unintuitively, more akin to single-agent than multi-agent reinforcement learning. This becomes clearer when considering a single agent acting as an 'overlord,' where the observation space is a combination of observations from all the individual agents. The actions taken by this overlord are combinations of actions chosen for each agent. Essentially, a single policy processes the combined observation and outputs the combined action.

(b) This example is a strong example of MARL, and because the agents utilize copies of a singular policy, this example is free from any of the types of heterogeneity described in the answer for question 1.1.

(c) The types of problems accurately described by the scenario provided in this part of the question appear to be a subset of MARL problems and a superset of HARL problems.

Many MARL algorithms allow the member policies to develop distinctly (e.g., [4]–[6]), but they are not optimized to facilitate the development of distinct policies.

Zhong et al. [3] formulate their series of HARL algorithms with optimizations intended to facilitate the development of distinct policies. One weakness of this formulation is that there is no guarantee that the multiple policies will not converge to a behaviorally indistinct set, similar to the concept of carcinization observed in evolutionary biology.

Thus, the resulting heterogeneity of the agents in these scenarios is not intrinsic but emergent. Whether the algorithm itself is labeled as MARL or HARL is distinguished by intent.

1.3 Referencing Centralized Training Decentralized Execution (CTDE) and Decentralized Training Decentralized Execution (DTDE) as employed by Li et al. in their FA2A paper [7], we see that CTDE is the most common format for Actor-Critic based MARL algorithms [4]–[8].

Li et al. [7] and Wen et al. [9] are the only papers I found that discuss the contrary implementation, DTDE. In both cases, the authors motivate DTDE with practical concerns, particularly the limitations of inter-agent communication in distributed systems. These considerations are important, but the relation to HARL remains the same as described in answer 1.2 (c).

2 Dr. Robbins' Questions

In addition to being included in the appendices of this exam, all code used to run the experiments is available on my github at https://github.com/bhosley/Specialty-Exam. It is written to be run on any ANT-Center VSCode server containers, but should work in a generic virtual environment. The specifications for the virtual machines used for this experiment are enumerated in table 1.

I recommended increasing the ratio of memory to CPU allocation for future experiments as the container was substantially closer to maximum utilization of memory than processing for the duration of the experiments run for this exam.

Setting	Value
Template	vscode-server
Image	reg.git.act3-ace.com/ace/hub/vscode-server:v0
Max CPUs	64
Req. CPUs	16
Memory	64 GB
GPU	None

Table 1: Container Settings

The github readme has a short guide for setting up the virtual environment in the ANT-Center for easy replication. The answers for this section are drawn from the experiment running script, also provided in appendix A of this document.

2.1 The Deep Q-network (DQN) implementation [10], [11] implemented in the RLlib framework [12] was used for each of the answers in this section. The default results output is readable by tensorboard, but I chose to use the wandb (Weights and Balances) api as well.

To perform this baseline experiment we can call the default script (appendix A). We can accomplish a 5 replication test by setting `--num-env-runners=5`, however, this overrides the default of 10, so we have chosen not to use it and to keep the default. We use the

```
python dqn_exp.py --sweep --num-samples=30 --num-env-runners=10
```

Round 1 - Bad Results:

In the first round of Parameter sweeping, a large number of training runs failed to converge on reasonable results. This was somewhat surprising, and when examining the parameters there was no immediately obvious pattern. Adding the average number of steps per episode to the Comparison (fig. 1) shows consistently shorter episodes for poor performance. Further investigation showed that the observation space assigned to the environment from the RLlib registered lunar lander environment was significantly different from the

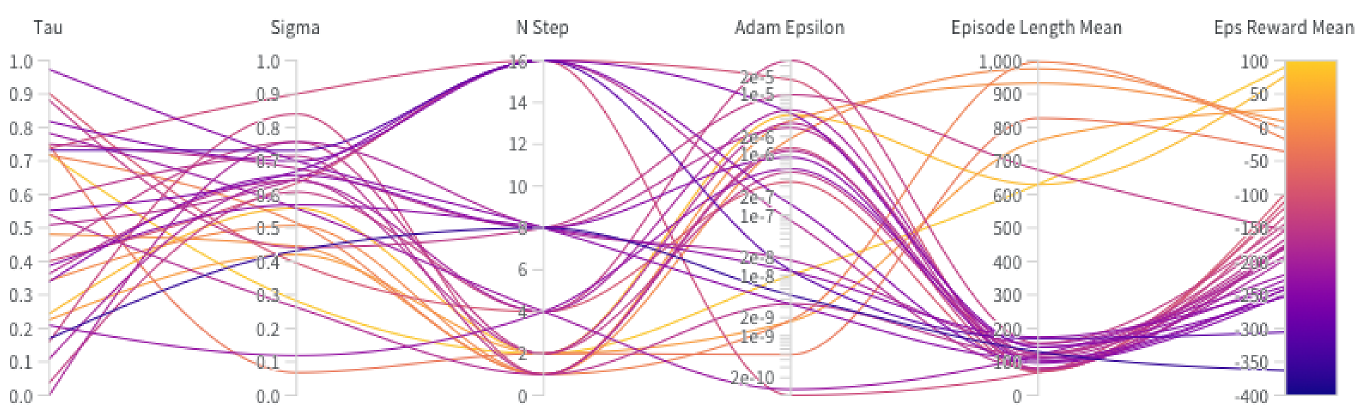


Figure 1: Lunar Lander Baseline Experiment 1 Hyperparameters

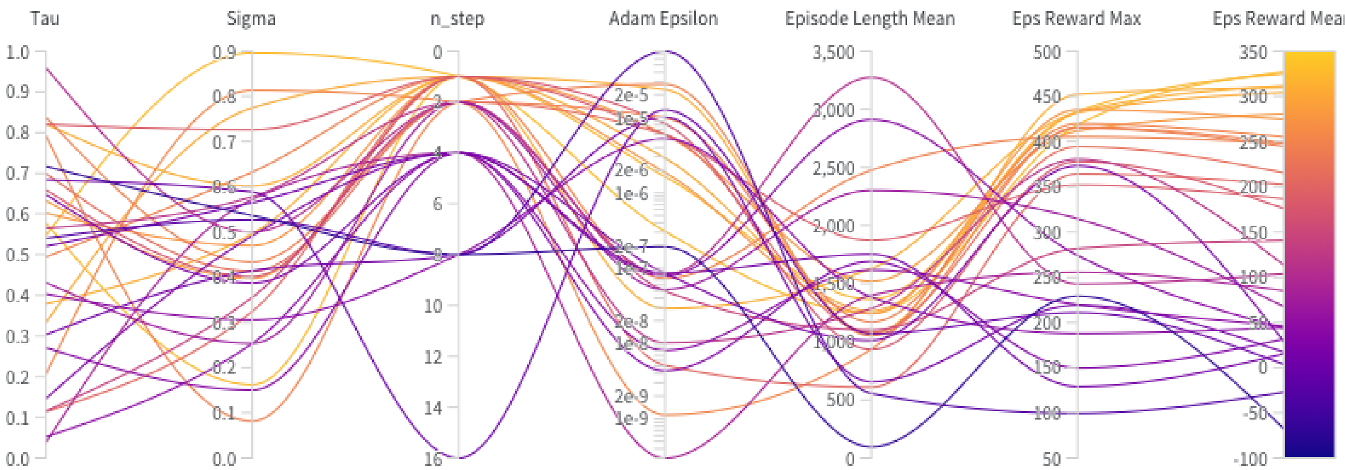


Figure 2: Lunar Lander Baseline Experiment 2 Hyperparameters

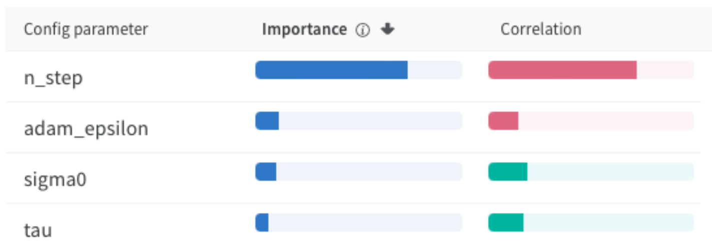


Figure 3: Baseline Experiment 2 Hyperparameter Importance

correct space. Further, manually registering the environment after importing it directly from the Farama maintained packages using Ray's environment registration function resulted in an outdated observation space, less prone to resulting in error, but still problematic.

Thus we gain two actionable insights. First, we adjust this experiment to use the custom Lunar Lander environment described in the next section, but set to one agent. Second, we identify an update that can be performed by a pull request to the maintainers of RLlib, a contribution to the open-source software I would like to attempt after the submission of this exam.

Round 2 - Baseline Parameter Sweep:

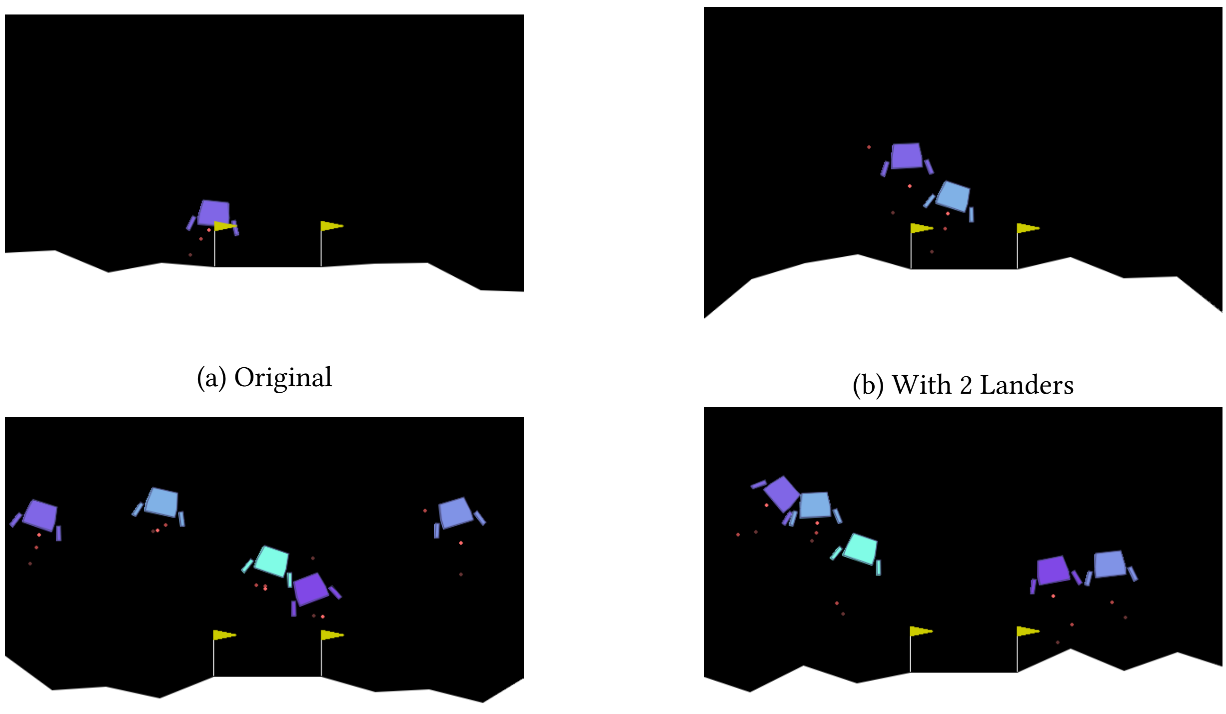
To evaluate the parameters of the sweep I elect to conduct a comparison with three different methods. I will use the same methods for the sweeps performed on the subsequent questions as well.

As above, the parallel coordinates graph provides a visual comparison of parameter values and a qualitative impression of the effects. Second, I use the wandb api Parameter importance and correlation graph, which provides an importance metric based on an algorithm from [13] that combines a tree-based regression and Spearman rank correlation; and the standard and Pearson correlation metric. Finally, for a more quantitative metric I produce an ANOVA table using second order interactions of the hyperparameters.

Figure 2 shows the parallel coordinates for this experiment's parameter sweep. Immediately this suggests that the n-step instances under-perform the single step instances. The factor importance (fig. 3) corroborates this. Finally, we execute the ANOVA in appendix C.1, and find that the σ parameter has a statistically significant effect on the final performance, particular as an interaction variable.

Multi-Agent Lunar Lander

To modify the Lunar Lander environment to support multiple agents it appeared to be possible to simply duplicate all of the sections that referenced the lander in the original code. However, to achieve cleaner code, and behavior more consistent with Farama's PettingZoo environments, I elected to rewrite the environment to be more consistent with object oriented programming principles.



(a) Original

(b) With 2 Landers

(c) With 5 Landers

(d) 5 Landers with Collision

Figure 4: Multi-Agent Lunar Lander

This trivialized the retention of the environmental physics, and made it very easy for the new environment to comply with both the AEC (Agent Environment Cycle; or sequential) and parallel execution methods used in the PettingZoo API. Further, this made it much easier to retain the heuristic solution from the original environment. Figure 4 shows several screen shots from the resulting environment when rendered.

While an AEC version of this custom environment was constructed, it was not used for this exam. AEC iterates through the list of agents, allowing the agent to select an action and steps the environment with that action before moving to the next agent, in a manner similar to a board game. Such an implement doesn't necessarily contribute value to the multi agent Lunar Lander sim, and in fact, causes problems if the physics of the environment are not adjusted as the number of landers is increased. Using the heuristic as a measure, the AEC environment becomes unsolvable with 3 or more landers. The effect of their actions must be scaled to overcome the wait time between their turns, otherwise they simple crash, unable to overcome gravity.

The parallel version of the environment was used for this exam, and required only one major assumption, centering around collisions between the landers. Lunar Lander uses the Box2D package to model the moon surface and lander. The most expedient method to address collisions using Box2D was to detect only contact with the lander's body. If any other object touches a lander's body it is considered a crash. As written, the Box2D contact detection does not distinguish between what the secondary object is, if the legs of the lander where included, contact with lunar surface would show as a crash. Thus I chose not to include the legs contact detection for collisions. The effective result is that if two lander's legs touch it is not treated as a crash, however, if one lander's legs touch another's body module it is treated as a crash and thus a failure.

In addition to being available on the associated github, the new environment code is included in appendix B

2.2 Formulate a Markov Game: The Markov game that represents the multi-agent version of this environment is an extension of the Markov Decision Process (MDP) that represents the original Lunar Lander environment. The formulation presented here will be used to describe the interactions in all of the questions that follow.

Agent: The agent is represented as a member of set of agents $n \in N$.

State Space: Let $\mathcal{S}$ be the state space. Then, $\mathcal{S} \equiv s^N$, where

$$s = \begin{cases} x \in [-2.5, 2.5] & \text{Position of } n \text{ in } x \\ y \in [-2.5, 2.5] & \text{Position of } n \text{ in } y \\ \vec{x} \in [-10, 10] & \text{Velocity of } n \text{ in } x \\ \vec{y} \in [-10, 10] & \text{Velocity of } n \text{ in } y \\ \omega \in [-2\pi, 2\pi] & \text{Angle of } n \\ \vec{\omega} \in [-10, 10] & \text{Angular Velocity of } n \\ \mathbb{I}(\text{leg } 1) \in \{0, 1\} & \text{Leg on ground} \\ \mathbb{I}(\text{leg } 2) \in \{0, 1\} & \text{Leg on ground} \end{cases}$$

Action Space: Let $\mathcal{A}$ be the action space. Then, $\mathcal{A} \equiv a^N$, where

$$a \in \begin{cases} 0 & \text{No-Op} \\ 1 & \text{Fire Left Engine} \\ 2 & \text{Fire Main Engine} \\ 3 & \text{Fire Right Engine} \end{cases}$$

Transition Probability: The transition probability for each agent n remains the same as the original environment,

$$P_n(s_{n,t+1} | s_{n,t}, a_{n,t}) \cong \begin{cases} \text{Dispersion} \sim U(-1, 1) \\ \text{Wind(Linear)} = \tanh(\sin(2kx) + \sin(\pi kx)) \\ \text{Wind(Rotate)} = \tanh(\sin(2kx) + \sin(\pi kx)) \end{cases}$$

Thus the transition probability for the game as a whole is

$$P(s_{t+1} | s_t, a_t) = \prod_{n \in N} P_n$$

Reward: The reward structure is similarly retained from the original Lunar Lander environment. Specifically, the reward at time t is defined as $r(t) = \sum_{n \in N} r_n(t)$ where

$$\begin{aligned} r_n(t) = & \\ & \pm 100 && \text{End-state, crash or land} \\ & + 10(s_{t,6} + s_{t,7}) && \text{leg(s) on ground} \\ & - a_n(t) \cdot [0, 0.03, 0.3, 0.03] && \text{thruster cost} \\ & - 100\sqrt{s_n(t)_0^2 + s_n(t)_1^2} && \text{Distance} \\ & - 100\sqrt{s_n(t)_2^2 + s_n(t)_3^2} && \text{Velocity} \\ & - 100|\omega_n(t)| && \text{Tilt} \end{aligned}$$

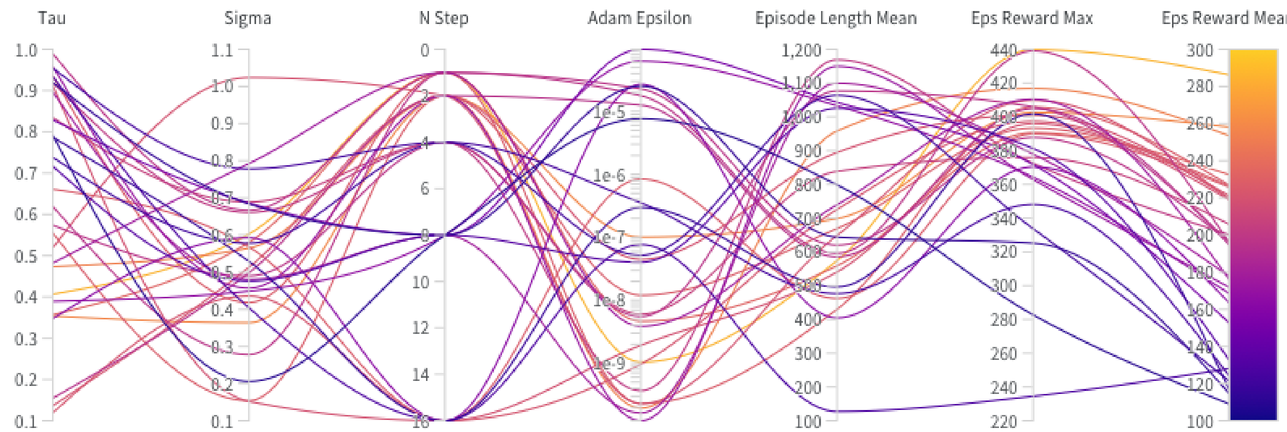


Figure 5: Single-agent Control Hyperparameter Sweep

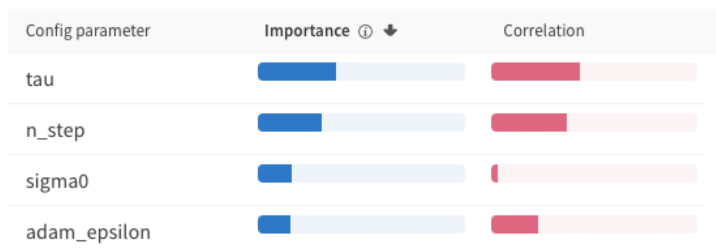


Figure 6: Single-agent Control Hyperparameter Importance

2.3 DQN Single-Agent Controller: This problem was approached using a wrapper class around the custom multi-agent environment to translate the between the multi-agent environment and single-agent policy. The proxy single-agent state/observation was formed by simply concatenating the state vectors of each agent.

The modified action space for this problem is $\mathcal{B} \equiv \mathcal{A}^N$. To relate the two action spaces for this problem I use the function,

$$b = \sum a_n \times \dim(a)^n.$$

Then to translate the modified action into the agent-action vector necessary for the multi-agent environment I use the following function,

$$\mathbf{a} = \{b / \dim(a)^n \mod \dim(a)\}_{n \in N}$$

which relies on $\dim(\mathcal{A}_n) = \dim(\mathcal{A}_m) \forall\, n, m \in N$, an assumption that I note as it is true for this problem, limits some generality. The policy can thus be represented as $\pi(b \big| [s_n]_{n \in N})$.

Finally, the experiment can be replicated using the command:

```
python dqn_exp.py --SA --sweep --num-samples=30 --num-env-runners=10
```

Figure 5 shows the results of the parameter sweep for this experiment. It appears from this information that there are no clear patterns suggesting what the optimal values are for the hyperparameters. Next, we consult fig. 6. From these metrics we do see some indication that the τ and n-step parameters may hold some importance for the final performance. And finally, the ANOVA table in appendix C.2, shows a statistical significance associated with the σ parameter.

I interpret the high statistical significance but extremely low correlation of the σ parameter to suggest that the mid-distribution value has a tendency to produce the best results. The superlative instance has a σ parameter near this 0.5, but overall, it appears that the algorithm is fairly resilient to Hyperparameter tuning.

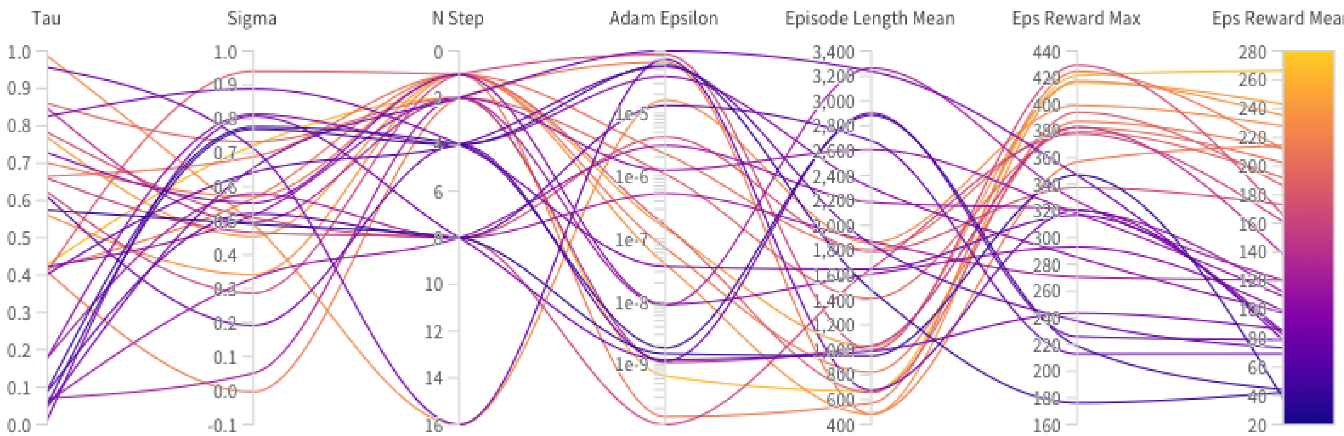


Figure 7: No Parameter Sharing Hyperparameter Sweep

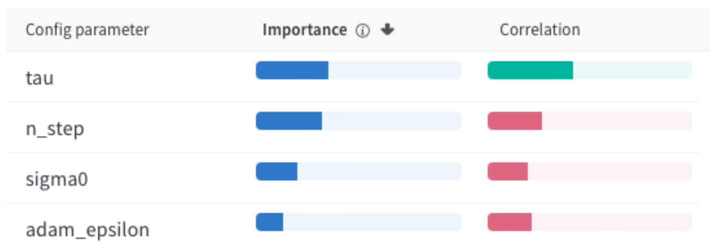


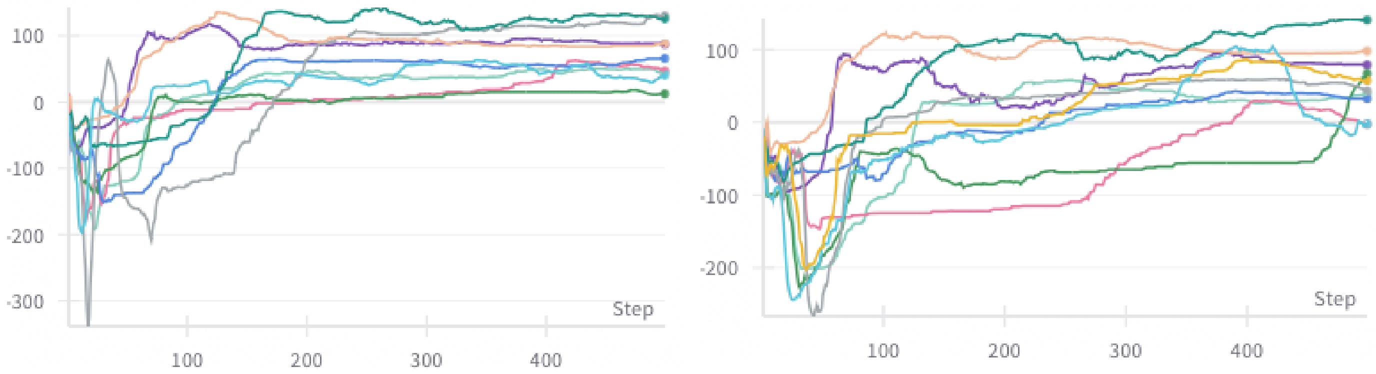
Figure 8: No Parameter Sharing Hyperparameter Importance

2.4 NoPS - No Parameter Sharing: This experiment uses a distinct policy for each agent, $\pi_n(a_n|s_n)$, and can be replicated using:

```
python src/dqn_exp.py --NoPS --sweep --num-samples=30 --num-env-runners=10
```

The parameter sweep results for this experiment can be seen in fig. 7. There doesn't appear to be any extremely clear or significant patterns from this parallel comparison.

There are, however, several observations of interest that are not directly concerning the hyperparameters. First, is a generally poor performance for instances that produce longer average episode length. Second, unlike the parameter sharing control, this consistently converged on a result that produced a positive reward in every case. I attribute this partially to sample size, but also note that this result is consistent with have two separate policies; that in order for the final mean episode reward to be negative, both policies would have to converge to poor behavior. This may also explain the more even distribution of average returns between the instances of this experiment.



(a) Agent 0 Reward

(b) Agent 1 Reward

Figure 9: No-parameter Sharing Learning Curves, average reward $\times$ training iteration

Next, fig. 8 shows the parameter importance metrics, which similarly suggests no strong relation between the Hyperparameter values and results, which is further corroborated in appendix C.3 which shows no statistically significant first or second order relationship between between the parameters and the final results. Finally, I present fig. 9 which shows the learning curves of the agents from 10 instances of this experiment. The resulting curves are consistent with pairs of agents that are learning differently.

2.5 FuPS - Full Parameter Sharing: This experiment uses a shared policy for each agent, $\pi(a_n|s_n)$, and can be replicated using:

```
python src/dqn_exp.py --FuPS --sweep --num-samples=30 --num-env-runners=10
```

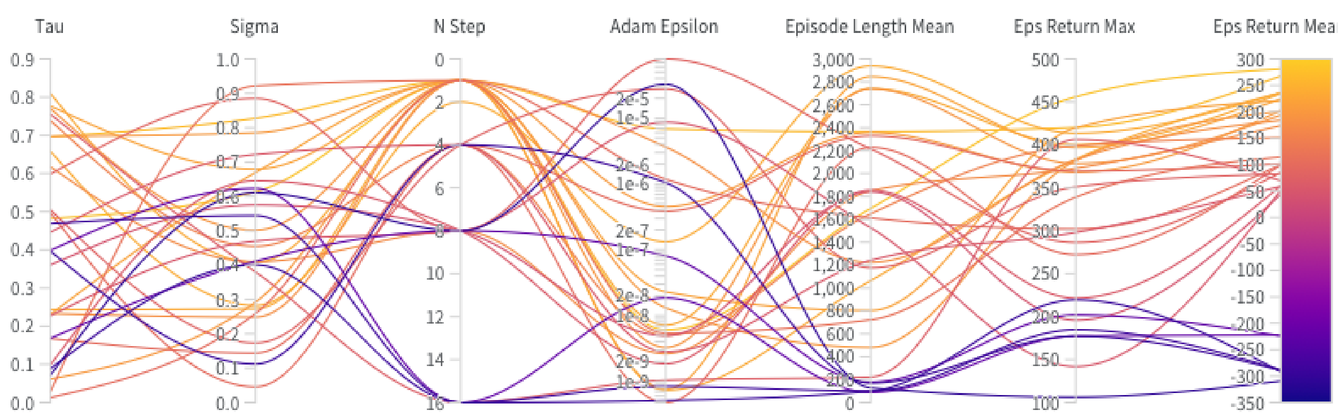


Figure 10: Full Parameter Sharing Hyperparameter Sweep

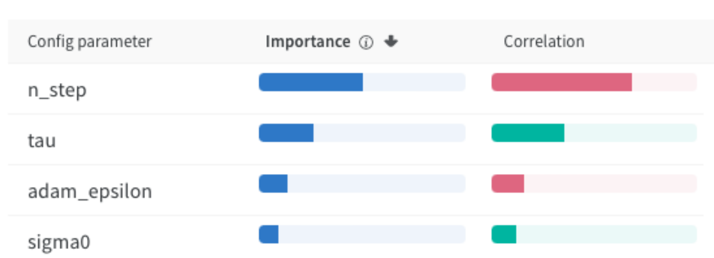


Figure 11: Full Parameter Sharing Hyperparameter Importance

The parameter sweep results for this experiment can be seen in fig. 10, once again it appears that there is a slightly higher performance result from a low or no n-step parameter. Further, when considering the additional compute required to execute the higher n instances, it is likely to be net negative. Figure 11 supports the high performance from lower n. Further, between the two figures, some support in favor of a $\tau > 0.5$. However, the ANOVA results (appendix C.4) suggest that these may not be statistically significant.

2.6 Results Comparisons:

Control	τ	σ	N-Step	Adam-ϵ
Single Agent	0.5	0.6	1	1e-5
NoPS	0.75	0.5	1	1e-7
FuPS	0.75	0.5	1	1e-8

Table 2: Hyperparameters used for control comparisons.

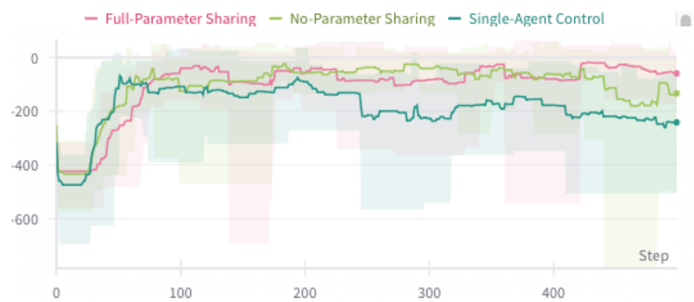
For the comparison I ran 10 instances of each algorithm using the parameters in table 2, which were values near those of the superlative performers in the previous sections. Some consideration was given to the default values of the DQN implementation. It may be noted that Mnih et al. [11] intended for their algorithm to be resilient and require minimal adjustments. The results seen in the previous sections seem to fit that intent, with exception to certain more extreme settings.

Figure 12 shows learning curve aggregates for each of the evaluated algorithms. From these results we see that the single-agent consistently under-performing the other algorithms, particularly, that after 250 episodes every instance's average was below the average of every instance of the other algorithms. Additionally, the variance in the average performance of the single-agent control algorithm was significantly greater than the other two algorithms. I attribute both of these observations to the larger action space and reduced ability to explore when using this control method.

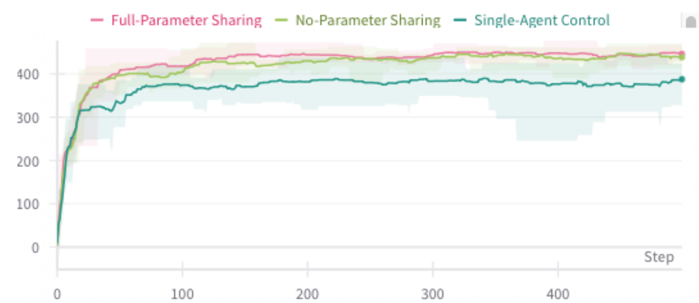
Figure 13 shows a measure of performance of the algorithms from the perspective of computational requirements. The CPU utilization for the single-agent control was by far the most consistent of the three, where the full-parameter sharing was generally higher while occasionally dropping to a utilization similar to the single-agent control. No-parameter sharing appears to have spent equal time between these two values. It is unclear from the experiments conducted for this exam why this would be the case. I would have expected that the NoPS instance would have a generally higher utilization than the others.

The memory utilization is about the same for each algorithm (fig. 14). Given that the algorithms use the same deep network architecture for their value functions, it reasonably follows that the memory usage would be similar, with small differences by the nature of the non-sharing networks are two separate networks it would not be surprising for those instances to have greater memory use, which appears to be the case by a small margin.

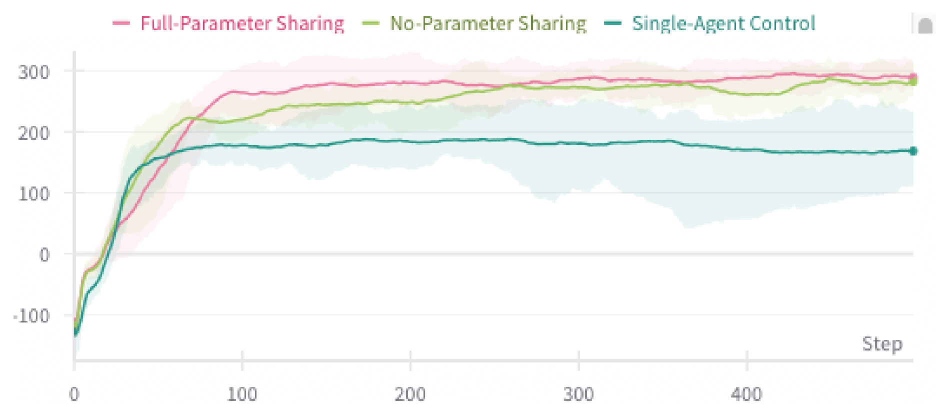
This same graph highlights perhaps the most significant advantage of the single agent control, which was an execution time almost half as long as the others. While this was not an advantage that I had considered prior to the experiment it seems that it would follow that the single agent controller uses the DQN once per step for each action and afterwards updates one network, where as the others would have to do so twice.



(a) Minimum Return



(b) Maximum Return



(c) Mean Return

Figure 12: Returns by Episodes

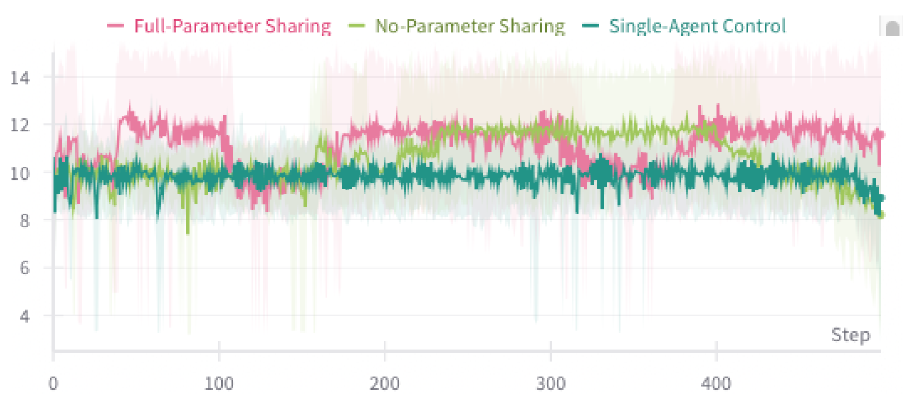


Figure 13: CPU Utilization

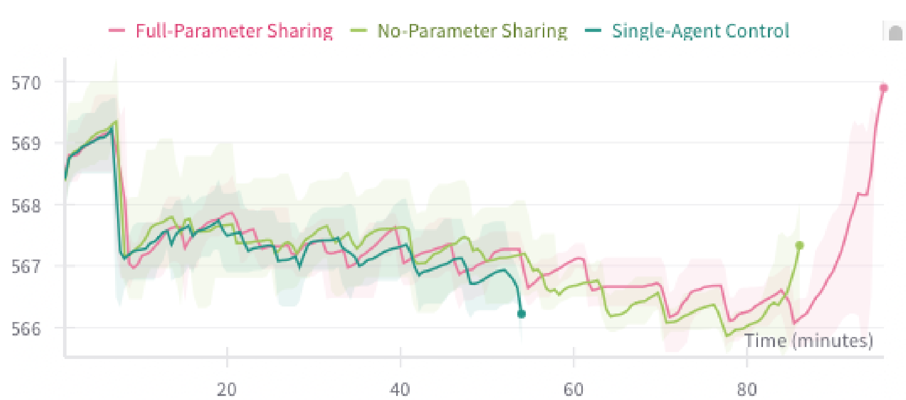


Figure 14: Memory Usage (MB)

3 Dr. Cox's Questions

3.1 I concur with the probable necessity of narrowing the scope of contribution one, particularly to maintain the feasibility of the proposed timeline.

(a) **Cooperative Tasks:** The SISL (Stanford Intelligent Systems Laboratory) environments are presented by Gupta et. al. in [14]. Two of the task in particular I believe to be usable, *Pursuit* and *Waterworld*. The latter of the two can be instantiated as a cooperate-competitive task. *Multiwalker* may be included, but that will be determined by whether or no I am able to get the task to function as expected with different numbers of walkers from the default 3. The two former environments are known to operate with a wide range of numbers of agents.

The MPE (Multi Particle Environment) is simple and has a variety of available tasks. However, among the manufacturer included tasks, *Simple Spread* is probably the only one suitable for the proposed evaluated. *Simple Spread* is a task that is essentially a zone-coverage/task-assignment type problem.

(b) **Competitive Tasks:** To maintain a greater focus on the adaptability and cooperation aspect of agent-groups I believe that it would be best to omit any pure competitive tasks. However, I intend to include cooperative-competitive tasks. As an example of this, let us take *Water World*; when the task is set such that a single agent can complete ('consume the food') it becomes a pure competitive environment, however, increasing that value is intended to produce emergent cooperation as the goal requires n agents to simultaneously activate the goal.

(c) **Algorithms:**

Base Algorithm	MARL	HARL	RLlib	RLlib-Contrib	SB3	Custom
PPO	MAPPO		☑			
PPO		HAPPO				☐
DDPG	MADDPG			☐		
DDPG		HADDPG				☐
TRPO	MATRPO				☐	
TRPO		HATRPO				☐
TD3	MATD3			☐		
TD3		HATD3				☐
A2C	MAA2C			☐		
A2C		HAA2C				☐

Table 3

Table 3 outlines the algorithms that I intend to try to utilize for contribution 1. They are broken up as to whether or not the original authors label the algorithm as a MARL of HARL algorithm. Next, they are labeled by a open checkbox corresponding to an open-source implementation that is available for the algorithm. Algorithms label RLlib are included in the base package, the -Contrib column are implementations that have not been fully tested and integrated into the main distribution of the package. They work within the *old-api* stack of RLlib, but may require some updates to function appropriately with the environments that I use. The SB3 column refers to Stable-Baselines 3, another well-established open-source library that implements several well known RL algorithms, which may be imported and potentially registered as trainable within the RLlib API. Notably, Trust Region Policy Optimization (TRPO) is the only algorithm that would potentially utilize this style of implementation.

The custom column represents the algorithms proposed by [3] which have been implemented in a library of their own, which is publicly available, but standalone. I appreciate that they made their code available, and provided instruction on hwo to reproduce their results, however, I do believe that there is benefit to conforming to the standards of a more popular framework in order to facilitate extension and

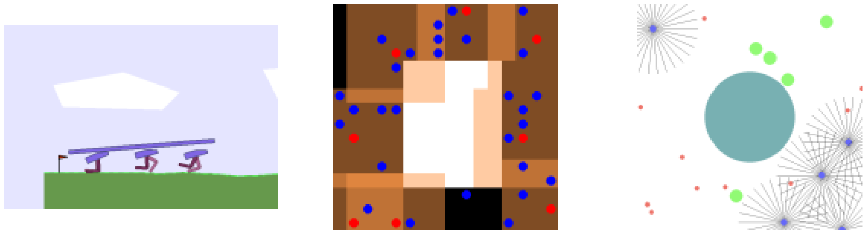


Figure 15: Multiwalker, pursuit, and waterworld environments

Environment	Source	RLlib Tested
Water World	SISL[14]	☑
Pursuit	SISL[14]	☑
Multiwalker	SISL[14]	☑
Multi-Particle Environment	Petting Zoo	☐
SMAX	[15]	☐
MAgent	[16]	☐

Table 4: Candidate Environments

application to other problems. The structure of their code is good, and most importantly, consistent, so I anticipate that it should carry over well. However, I ran into time constraints, and did not have enough time to move the code over and test it appropriately for this exam. I intend to do so in the time immediately following.

The check mark indicates algorithms that have been thoroughly tested and function without issue for arbitrary environments (provided that the environment is compliant with the API).

(d) **Environments:** Table 4 lists the environments that I would like to use as metrics for the algorithms. As alluded to in parts (a) and (b), the key feature that I have used in picking these environments is that the variability in the number of agents that the task supports.

The environments that I have confirmed to register without issue with the RLlib API are the examples that I have described in parts (a) and (b) of this exam question. However, there are other environments that I would like to attempt to implement as well. At risk of being overly repetitive, a major advantage of working within an established API provides me the flexibility to more easily extend the experiments to additional environments once they are compliant with the API, typically via wrapper. Moreover, this provides some flexibility with regard to time and size of contribution 1.

(e) **League:** Evaluation of the league format is definitely something that is becoming clearly a scope challenge, particularly with the timeline concerns. As I have been working on this exam, and exploring the framework that I have been using for the experiments, I have been forced to reflect more deeply upon the individual components of the models. I have become convinced that league play is something that should be evaluated later, with closer proximity to curriculum design than where I am currently.

3.2 The following section is the *mini-paper*. The biggest obstacle currently is, of course, time. In the interest of time, the experiment presented does not have the replications that I would deem necessary, but should serve as a proof of functionality on ACE-hub.

While working on this exam I have discovered a potential limitation with using ACE-hub. Long running scripts appear to generally proceed without incident on the containers when disconnected; however, when utilizing an API that opens a port to connect to an external service, disconnecting from the web-based user interface appears to pause execution. At the time of submission of this exam I have one workaround, which is to forego exporting the data during the experiment and instead to export the results manually afterward.

Mini-Paper

Introduction

Multi-agent reinforcement learning (MARL) has garnered significant attention in recent years due to its potential to solve complex, collaborative, and competitive tasks across various domains. The extension to heterogeneous-agent reinforcement learning (HARL) further broadens the scope by incorporating diverse agents with differing capabilities and roles. As the field progresses, a critical area of research lies in understanding how different MARL and HARL algorithms perform under varying conditions, such as changes in the number of participating agents.

This paper presents an experimental study that evaluates several state-of-the-art MARL and HARL algorithms under dynamic conditions where the number of agents changes between training and deployment. The primary aim is to establish a comparative baseline for the performance and robustness of these algorithms, contributing to the body of knowledge on their practical applicability and effectiveness. By replicating and contrasting the results reported by the original authors of these algorithms, this study seeks to either corroborate or challenge existing findings, thereby enhancing the reliability of performance benchmarks in MARL and HARL.

Furthermore, this experiment investigates the resource costs associated with the initial training phases of these algorithms. Understanding the computational and time resources required to achieve optimal performance under standard conditions is crucial for deploying these algorithms in real-world applications. Additionally, we assess the resource implications of adapting the algorithms to maintain benchmark performance when the number of agents changes during deployment. This dual focus on initial training and adaptive performance evaluation provides a comprehensive view of the efficiency and scalability of current MARL and HARL approaches.

Through this examination, we aim to highlight the strengths and weaknesses of the tested algorithms, offering insights into their suitability for different multi-agent environments. The findings from this study are expected to inform future research and development in the field, guiding improvements in algorithm design and deployment strategies.

Related Work

The field of multi-agent reinforcement learning (MARL) and its extension to heterogeneous-agent reinforcement learning (HARL) has seen significant advancements over the past decade. This section reviews the key literature relevant to our study,

DeepMind's development of AlphaGo [17] and subsequent iterations, AlphaGo Zero [18] and AlphaStar [19], demonstrated the power of deep reinforcement learning in multi-agent environments. These models, particularly AlphaStar, which operates in the complex, dynamic environment of StarCraft II, highlight the scalability and effectiveness of MARL algorithms. They provide an example of what is achievable, albeit with substantial resources of a scale unavailable to the vast majority of researchers or other organizations. In response, there have been numerous efforts to improve efficiency at various levels of training.

Smit et al. [20] focused on leveraging scalability to reduce training requirements. They used a simulated football environment which, at full scale, uses two teams of 11 agents. They explored extensions of the PPO algorithm to attempt to produce an effective team of 11 agents, while training only 4. Ultimately they were unable to achieve the performance that they sought. In their work they did not address why PPO was selected over any other algorithm.

Algorithms:

Lowe et al. [6] observed weaknesses in Q-learning and policy gradient methods when applied to multi-agent settings. They proposed an algorithm that they called multi-agent deep-deterministic policy-gradient (MADDPG), which extends a single agent DDPG [21] with a shared critic.

Multi-agent Twin Delayed Deep Deterministic policy gradient (MATD3) [22] extends TD3 [23], also using a shared critic.

Multi-agent Trust Region Policy Optimization (MATRPO) [24] extends TRPO [25] using a shared likelihood ratio on action optimality that has the advantage of preserving privacy of each agent's own policy.

Multi-agent Proximal Policy Optimization (MAPPO) [26] extends PPO [27] through shared parameters.

Importance Weighted Actor-Learner Architecture (IMPALA) [28] is an algorithm with a central learner but with highly decentralized execution, that supports homogeneous multi-agent applications from the original implementation.

Zhong et al. [3] proposed a suite of algorithms, which extend MADDPG, MATD3, MATRPO, and MAPPO, to be more flexible with the agents that they are applied to. They sought to improve the efficiency of training agents capable of cooperation through changes to the original algorithms that emphasized development of distinct behavior.

They called their modified MARL algorithms, heterogeneous-agent reinforcement learning (HARL). Though the agents themselves are not heterogeneous, the algorithms are written to encourage the development of heterogeneous policies and behavior. Along with their paper, Zhong et al. [3] do provide the code that they used to implement their algorithms and perform their experiments. Their experiments evaluated each of their algorithms against the algorithm that it was based when trained in six common baseline environments.

Environments:

Gupta et al. [14] published a set of cooperative tasks for evaluating MARL algorithms. Collectively they are sometimes referred to as the SISL (Stanford Intelligent Systems Laboratory) environments. The environments included in the SISL benchmarks are called Multiwalker, Pursuit, and Waterworld. Each provide a distinct task that requires cooperation and they have parameters that allow the number of agents instantiated to be changed.

Other environments to be described as they are implemented...

Methodology

This section describes the experimental methodology used to evaluate the selected algorithms.

Fixed Agent Count Training: Each algorithm was trained with several different fixed numbers of agents. The training process followed the protocols and hyperparameters recommended by the authors of the respective algorithms. Training was conducted over 200 episodes.

Performance metrics such as cumulative rewards, wall-clock training time, CPU utilization, and memory utilization are collected during this training period, and will relate directly to previously reported metrics claimed in other works.

Evaluation Under Altered Conditions: After initial training, the agents are evaluated on the task with and altered team size. Specifically, we tested each model with both fewer and more agents than were present during training. The variations included incremental increases and decreases in the agent count to observe performance trends.

Adaptive Training: During this step we resume the training of the team of agents under the new environmental conditions. In doing so we seek to evaluate both the feasibility of this method of retraining, and to determine if there are resource benefits when compared to training an entirely new set of agents with the expected team size.

Waterworld: This environment implements a task that simulates small organisms that must cooperate to gather food items and avoid poisoned items. Each agent has a range limited, 212-dimension observation space. By default they are rewarded $+10$ for successful cooperation and consumption of a food item, and -1 for colliding with a poisoned item. For reward shaping a $+0.1$ is given to each agent when they collide with a food item, independent of if they are able to consume it.

The principle variables that we use in this experiment are the number of agents in the environment, and the number of agents that are need to cooperatively consume a food item.

Multiwalker: This environment is an extension of the Farama gymnasium bipedal walker. It consists of several bipedal walkers with a long box placed upon their head. The action space for the walkers consists

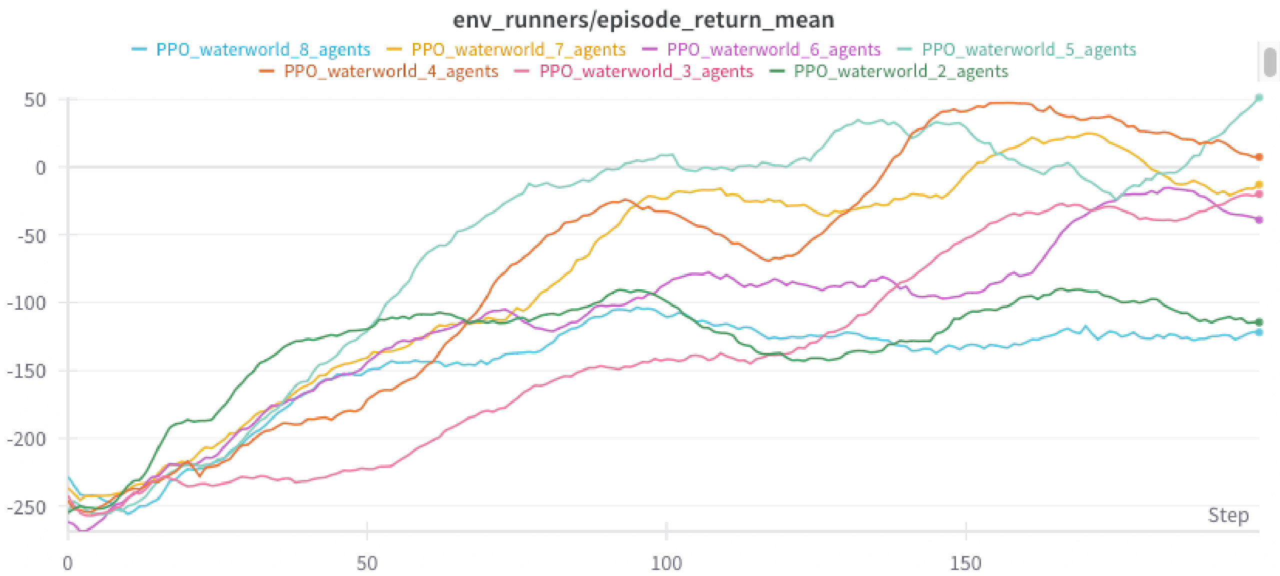


Figure 16: Mean Training Episode Return for Waterworld (Coop=2)

of changing the angles of their leg joints. Each walker has an observation space that is 32-dimensional and represents a noisy lidar reading of the area in front of the walker. The range is limited enough that they are only able to observe up to two walkers in front of themselves. Dropping the package nets a reward of -100, and moving the package forward rewards $+1$.

Although it was not emphasized by the authors, the walkers themselves are homogeneous in capability, but their positioning does create heterogeneous observation spaces, and potentially functionality; particularly for the first two walkers compared to the rest.

This is taken into account during the evaluation with differing team size, when the agents have distinct policies. Because three is the minimum number of walkers that the environment supports, the first two always remain the first two. The change in team size is achieved by randomly adding copies selected from the third or greater agent, to increase the team size; or randomly omitting an agent third or greater to reduce the team size.

Early Results

Waterworld:

Preliminary results (fig. 16) show that unsurprisingly, PPO does reliably converge on increasingly effective policies. Additionally, it also appears that there is an inflection point at which increasing the number of agents reduces the total return. This is too small of a sample to draw meaningful conclusions but it does suggest that we will want to be aware that such an inflection point may exist. For the environment where the cooperation parameter is set to 2, it appears that somewhere around 4 or 5 agents achieves maximum performance. Below this point it is likely that the agents are too few to effectively accomplish the task, and above this point the agents begin to compete and that there are too few objectives, while increasing the collisions that result in negative returns.

Multiwalker:

The first runs of Multiwalker returns (fig. 17) show earlier and faster improvements with fewer agents, but over a longer training period, the greater number of agents net a higher reward. It should be noted that the positive return for this task is the same for each agent, but it is added to each agent's individual score. The negative for dropping the box is also given to all agents. The penalties for an agent falling or moving its head to an extreme angle are added only to that agent's score.

Figure 17: Mean Training Episode Return for Multiwalker

Future Work

After this exam has been turned in, work will continue on the subject of this mini-paper. The intended contribution will include a framework to allow others to replicate the results of this paper, and be built in a manner that should be easier for others to use the constituent parts and tools that are currently being developed. Additionally, none of the HARL algorithms have been implemented in either of the most common frameworks, and thus implementations built for this experiment will be the subject of a pull request to the contribution branch of RLlib.

References

[1] J. A. Calvo and I. Dusparic, “Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control,” *AICS*, pp. 2–13, Dec. 2018.

[2] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. “Dota 2 with Large Scale Deep Reinforcement Learning.” arXiv: `1912.06680 [cs, stat]`. (Dec. 13, 2019), [Online]. Available: `http://arxiv.org/abs/1912.06680` (visited on 03/15/2024), pre-published.

[3] Y. Zhong, J. G. Kuba, X. Feng, S. Hu, J. Ji, and Y. Yang, “Heterogeneous-Agent Reinforcement Learning,” *Journal of Machine Learning Research*, vol. 25, no. 32, pp. 1–67, 2024. [Online]. Available: `http://jmlr.org/papers/v25/23-0488.html`.

[4] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. “Counterfactual Multi-Agent Policy Gradients.” arXiv: `1705.08926 [cs]`. (Dec. 14, 2017), [Online]. Available: `http://arxiv.org/abs/1705.08926` (visited on 04/04/2024), pre-published.

[5] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. “QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning.” arXiv: `1803.11485 [cs, stat]`. (Jun. 6, 2018), [Online]. Available: `http://arxiv.org/abs/1803.11485` (visited on 04/04/2024), pre-published.

[6] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.” arXiv: `1706.02275 [cs]`. (Mar. 14, 2020), [Online]. Available: `http://arxiv.org/abs/1706.02275` (visited on 02/12/2023), pre-published.

[7] W. Li, B. Jin, X. Wang, J. Yan, and H. Zha, “F2A2: Flexible Fully-decentralized Approximate Actor-critic for Cooperative Multi-agent Reinforcement Learning,” *Journal of Machine Learning Research*, vol. 24, no. 178, pp. 1–75, 2023, ISSN: 1533-7928. [Online]. Available: `http://jmlr.org/papers/v24/20-700.html` (visited on 05/30/2024).

[8] Y. Zhou, S. Liu, Y. Qing, K. Chen, T. Zheng, Y. Huang, J. Song, and M. Song. “Is Centralized Training with Decentralized Execution Framework Centralized Enough for MARL?” arXiv: `2305.17352 [cs]`. (May 26, 2023), [Online]. Available: `http://arxiv.org/abs/2305.17352` (visited on 06/25/2024), pre-published.

[9] G. Wen, J. Fu, P. Dai, and J. Zhou, “DTDE: A new cooperative multi-agent reinforcement learning framework,” *The Innovation*, vol. 2, p. 100 162, Sep. 1, 2021. DOI: `10.1016/j.xinn.2021.100162`.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. “Playing Atari with Deep Reinforcement Learning.” arXiv: `1312.5602 [cs]`. (Dec. 19, 2013), [Online]. Available: `http://arxiv.org/abs/1312.5602` (visited on 07/21/2024), pre-published.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 26, 2015, ISSN: 0028-0836, 1476-4687. DOI: `10.1038/nature14236`. [Online]. Available: `https://www.nature.com/articles/nature14236` (visited on 07/21/2024).

[12] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” 2018. arXiv: `1807.05118`.

[13] J. Howard, S. Gugger, S. Chintala, and a. O. M. C. Safari, *Deep Learning for Coders with Fastai and PyTorch: AI Applications without a PhD*. O’Reilly Media, Incorporated, 2020, ISBN: 978-1-4920-4552-6. [Online]. Available: `https://books.google.com/books?id=xd6LxgEACAAJ`.

[14] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, Springer, 2017, pp. 66–83.

[15] A. Rutherford, B. Ellis, M. Gallici, J. Cook, A. Lupu, G. Ingvarsson, T. Willi, A. Khan, C. S. de Witt, A. Souly, S. Bandyopadhyay, M. Samvelyan, M. Jiang, R. T. Lange, S. Whiteson, B. Lacerda, N. Hawes, T. Rocktaschel, C. Lu, and J. N. Foerster. "JaxMARL: Multi-Agent RL Environments in JAX." arXiv: `2311.10090 [cs]`. (Dec. 19, 2023), [Online]. Available: `http://arxiv.org/abs/2311.10090` (visited on 06/04/2024), pre-published.

[16] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu. "MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence." arXiv: `1712.00600 [cs]`. (Dec. 2, 2017), [Online]. Available: `http://arxiv.org/abs/1712.00600` (visited on 06/25/2024), pre-published.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 1476-4687. DOI: `10.1038/nature16961`. [Online]. Available: `https://www.nature.com/articles/nature16961` (visited on 03/15/2024).

[18] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, ISSN: 1476-4687. DOI: `10.1038/nature24270`. [Online]. Available: `https://www.nature.com/articles/nature24270` (visited on 03/15/2024).

[19] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 14, 2019, ISSN: 0028-0836, 1476-4687. DOI: `10.1038/s41586-019-1724-z`. [Online]. Available: `https://www.nature.com/articles/s41586-019-1724-z` (visited on 12/24/2023).

[20] A. Smit, H. A. Engelbrecht, W. Brink, and A. Pretorius, "Scaling multi-agent reinforcement learning to full 11 versus 11 simulated robotic football," *Autonomous Agents and Multi-Agent Systems*, vol. 37, no. 1, p. 20, Mar. 24, 2023, ISSN: 1573-7454. DOI: `10.1007/s10458-023-09603-y`. [Online]. Available: `https://doi.org/10.1007/s10458-023-09603-y` (visited on 03/16/2024).

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning." arXiv: `1509.02971 [cs, stat]`. (Jul. 5, 2019), [Online]. Available: `http://arxiv.org/abs/1509.02971` (visited on 05/25/2024), pre-published.

[22] J. Ackermann, V. Gabler, T. Osa, and M. Sugiyama. "Reducing Overestimation Bias in Multi-Agent Domains Using Double Centralized Critics," arXiv.org. (Oct. 3, 2019), [Online]. Available: `https://arxiv.org/abs/1910.01465v2` (visited on 05/26/2024).

[23] S. Fujimoto, H. van Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods." arXiv: `1802.09477 [cs, stat]`. (Oct. 22, 2018), [Online]. Available: `http://arxiv.org/abs/1802.09477` (visited on 05/25/2024), pre-published.

[24] H. Li and H. He. "Multi-Agent Trust Region Policy Optimization." arXiv: `2010.07916 [cs]`. (Aug. 4, 2023), [Online]. Available: `http://arxiv.org/abs/2010.07916` (visited on 05/30/2024), pre-published.

[25] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization." arXiv: 1502.05477 [cs]. (Apr. 20, 2017), [Online]. Available: http://arxiv.org/abs/1502.05477 (visited on 05/25/2024), pre-published.

[26] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. "The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games." arXiv: 2103.01955 [cs]. (Nov. 4, 2022), [Online]. Available: http://arxiv.org/abs/2103.01955 (visited on 05/25/2024), pre-published.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms." arXiv: 1707.06347 [cs]. (Aug. 28, 2017), [Online]. Available: http://arxiv.org/abs/1707.06347 (visited on 05/25/2024), pre-published.

[28] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." arXiv: 1802.01561 [cs]. (Jun. 28, 2018), [Online]. Available: http://arxiv.org/abs/1802.01561 (visited on 06/26/2024), pre-published.

A Experiment Running Script

```python
"""Script for Dr. Robbins Questions
Establishing a baseline implementation of Single Agent Lunar Lander
with a DQN.

See: https://gymnasium.farama.org/environments/box2d/lunar_lander/
for more details on the environment.

How to run this script
----------------------
`python [script_name].py`

For debugging, use the following additional command line options
`--no-tune --num-env-runners=0`

For logging to your WandB account, use:
`--wandb-key=[your WandB API key] --wandb-project=[some project name]
--wandb-run-name=[optional: WandB run name (within the defined project)]`

"""

import numpy as np
from gymnasium import Wrapper, spaces
from argparse import ArgumentParser

from ray import tune
from ray.rllib.core.rl_module.rl_module import SingleAgentRLModuleSpec
from ray.rllib.core.rl_module.marl_module import MultiAgentRLModuleSpec
from ray.rllib.env.wrappers.pettingzoo_env import ParallelPettingZooEnv
from ray.rllib.utils.test_utils import (add_rllib_example_script_args,
                                        run_rllib_example_script_experiment)
from ray.tune.registry import get_trainable_cls, register_env

import multi_lander


parser = add_rllib_example_script_args(
    ArgumentParser(conflict_handler='resolve'), # Resolve for num_agents
    default_reward=600.0,
    default_iters=500, #100, #200
    default_timesteps=1000000, #10000, #100000
)
parser.add_argument(
    "--control",
    type=str,
    choices=["Baseline", "SA", "NoPS", "FuPS"],
    default="Baseline",
    help="The controller method."
    "`Baseline`: Original Lunar Lander with single agent and lander"
    "`SA`: A single agent controls all of the landers."
    "`NoPS`: No parameter sharing between the agents."
    "`FuPS`: Full parameter sharing between the agents.",
)
parser.add_argument(
    "--SA", action="store_true",
    help="`SA`: Creates a single agent for controlling all of the agents.",
)
parser.add_argument(
    "--NoPS", action="store_true",
```

```
    help="`NoPS`: No parameter sharing between the agents.",
)
parser.add_argument(
    "--FuPS", action="store_true",
    help="`FuPS`: Full parameter sharing between the agents.",
)
parser.add_argument(
    "--num-agents", type=int, default=2,
    help="The number of agents",
)
parser.add_argument(
    "--sweep", action="store_true",
    help="Perform a parameter sweep instead of using tune",
)


class CustomWrapper(ParallelPettingZooEnv):
    """Wraps a Parallel Petting Zoo Environment for RLlib

    This wrapper is necessary to interface with rllib's workers.
    There is a problem caused when the one agent terminates before
    the other, which seems to pass inconsistent length episode
    trajectories to the rollout worker(s).

    It also adds the necessary `__all__` key to the done dictionaries.
    """
    def step(self, action_dict) -> tuple:
        obs, rew, terminateds, truncateds, info = self.par_env.step(action_dict)

        active = [a_id for a_id,term in terminateds.items() if term==False]

        obs_s = {a_id: obs.get(a_id) for a_id in active}
        rew_s = {a_id: rew.get(a_id) for a_id in active}
        terminateds = {a_id: terminateds.get(a_id) for a_id in active}
        truncateds = {a_id: truncateds.get(a_id) for a_id in active}

        terminateds["__all__"] = all(terminateds.values())
        truncateds["__all__"] = all(truncateds.values())
        return obs_s, rew_s, terminateds, truncateds, info

# Registry is necessary for functional passing later.
register_env("ma-lander", lambda _: CustomWrapper(multi_lander.Parallel_Env()))


class SingleAgentWrapper(Wrapper):
    """Wraps a parallel multi-agent environment for single-agent control.

    *It is currently limited to agents with discrete action spaces and
    box observation spaces with identical lengths.

    This will wrap environments with an arbitrary number of agents,
    however, the action space scales exponentially by the number of
    agents and is likely to be the limiting factor for performance.
    """
    def __init__(self,*args):
        super().__init__(*args)
        # Concat all action spaces into a single action space
        self.action_space = spaces.Discrete(np.prod(
                            [list(self.env.action_spaces.values())[i].n
                            for i in range(len(self.env.observation_spaces))]))
```

```
        self._act_n = list(self.env.action_spaces.values())[0].n
        # Concat all obs spaces into a single obs space
        low,high = [],[]
        for i in range(len(self.env.observation_spaces)):
            low.extend(list(self.env.observation_spaces.values())[i].low)
            high.extend(list(self.env.observation_spaces.values())[i].high)
        self.observation_space = spaces.Box(np.array(low), np.array(high))

    def observe(self) -> tuple:
        # Concat all obs values into a single obs tuple
        #return sum(self.env.observe().values(),[]) # By monoid
        return np.array(list(self.env.observe().values())).flatten()

    def observation(self) -> tuple:
        return self.observe()

    def reset(self,*, seed:int | None=None, options:dict | None=None) -> tuple:
        self.env.reset(seed=seed, options=options)
        return self.observe(), {}

    def step(self, action:int) -> tuple:
        action_list = {a: int(action/(self._act_n**i))%self._act_n
                        for i,a in enumerate(self.agents)}
        obss, rews, terminations, _, _ = self.env.step(action_list)
        obs = self.observe()
        # Calculate reward, but drop terminated agent, else get exploding values
        reward = sum(np.array(rews.values()) *
                     np.invert(list(terminations.values())))
        term = (self.env.game_over or all(terminations.values()))
        if (term or np.any(obs < self.observation_space.low) or
                    np.any(obs > self.observation_space.high)):
            term = True
            obs = np.clip(obs, self.observation_space.low,
                          self.observation_space.high)
        return obs, reward, term, False, {}

# Registry is necessary for functional passing later.
register_env("sa-lander",
    lambda _: SingleAgentWrapper(multi_lander.Parallel_Env()))


if __name__ == "__main__":
    args = parser.parse_args()

    # Shared config settings for all experiments
    base_config = (
        get_trainable_cls("DQN")
        .get_default_config()
        .framework('torch')
    )

    # Set config for experiment if using Single Agent Control
    if args.control == 'NoPS' or args.SA:
        config = (
            base_config
            .environment(
                "sa-lander",
                env_config={"num_landers": args.num_agents}
            )
        )
```

```python
    # Set config for experiment if using No-Parameter Sharing
    elif args.control == 'NoPS' or args.NoPS:
        policies = {"lander_" + str(i) for i in range(args.num_agents)}
        config = (
            base_config
            .multi_agent(
                policies=policies, # 1:1 map from AgentID to ModuleID.
                policy_mapping_fn=(lambda aid, *args, **kwargs: aid),
            )
            .rl_module(
                rl_module_spec=MultiAgentRLModuleSpec(
                   module_specs={p:SingleAgentRLModuleSpec() for p in policies},
                ),
            )
            .environment(
                "ma-lander",
                env_config={"num_landers": args.num_agents}
            )
        )
    # Set config for experiment if using Full-Parameter Sharing
    elif args.control == 'FuPS' or args.FuPS:
        config = (
            base_config
            .multi_agent(
                policies={"p0"}, # All agents map to the same policy.
                policy_mapping_fn=(lambda aid, *args, **kwargs: "p0"),
            )
            .rl_module(
                rl_module_spec=MultiAgentRLModuleSpec(
                    module_specs={"p0": SingleAgentRLModuleSpec()},
                ),
            )
            .environment(
                "ma-lander",
                env_config={"num_landers": args.num_agents}
            )
        )
    # Default, original lunar lander with one agent and lander
    else:
        config = (
        base_config
        .environment(
            # env="lunar-lander"
            "ma-lander",
            env_config={"num_landers": 1}
        )
    )

    # Parameter sweep settings
    if args.sweep:
        param_space = {
            "adam_epsilon": tune.loguniform(1e-4, 1e-10), # 1e-8
            "sigma0": tune.randn(0.5, 0.2), # 0.5
            "n_step": tune.choice([2**i for i in range(5)]), # 1
            # Update the target by \tau * policy + (1-\tau) * target_policy.
            "tau": tune.uniform(0.0,1.0), # 1.0,
            #epsilon = [(0, 1.0), (10000, 0.05)]    [(step,epsilon),...]
            # -> 1.0 at beginning, decreases to 0.05 over 10k steps
        }
        config = config.training(**param_space)
```

```
        # Use Param sweep, not tune
        #args.no_tune = True

    # Call experiment runner
    run_rllib_example_script_experiment(config, args)
```

B Multi-Agent Lunar Lander

```python
import math
from typing import TYPE_CHECKING, Optional
import numpy as np

import gymnasium as gym
from gymnasium import spaces, logger
from gymnasium.error import DependencyNotInstalled
from gymnasium.utils import EzPickle

from pettingzoo import AECEnv
from pettingzoo.utils import agent_selector

try:
    import Box2D
    from Box2D.b2 import (
        circleShape,
        contactListener,
        edgeShape,
        fixtureDef,
        polygonShape,
        revoluteJointDef,
    )
except ImportError as e:
    raise DependencyNotInstalled(
        'Box2D is not installed, you can install it by run `pip install swig` '
        + 'followed by `pip install "gymnasium[box2d]"`'
    ) from e

if TYPE_CHECKING:
    import pygame

FPS = 50
SCALE = 30.0  # affects how fast the game is, forces should be adjusted as well
MAIN_ENGINE_POWER = 13.0
SIDE_ENGINE_POWER = 0.6

INITIAL_RANDOM = 1000.0  # Set 1500 to make game harder

LANDER_POLY = [(-14, +17), (-17,0), (-17, -10), (+17, -10), (+17,0), (+14, +17)]
LEG_AWAY = 20
LEG_DOWN = 18
LEG_W, LEG_H = 2, 8
LEG_SPRING_TORQUE = 40

SIDE_ENGINE_HEIGHT = 14
SIDE_ENGINE_AWAY = 12
MAIN_ENGINE_Y_LOCATION = 4 # The Y loc of the main engine on the lander body.

VIEWPORT_W = 600
VIEWPORT_H = 400


class ContactDetector(contactListener):
    def __init__(self, env):
        contactListener.__init__(self)
        self.env = env

    def BeginContact(self, contact):
```

```python
        for lander in self.env.landers:
            if (
                lander == contact.fixtureA.body
                or lander == contact.fixtureB.body
            ):
                self.env.game_over = True
        for leg in self.env.legs:
            if leg in [contact.fixtureA.body, contact.fixtureB.body]:
                leg.ground_contact = True

    def EndContact(self, contact):
        for leg in self.env.legs:
            if leg in [contact.fixtureA.body, contact.fixtureB.body]:
                leg.ground_contact = False


class SequentialEnv(AECEnv, EzPickle):
    r"""
    This is a rewrite of the Farama foundation Lunar Lander environment from:
        https://gymnasium.farama.org/environments/box2d/lunar_lander/
    Adapted for a AEC type petting zoo environment.
    The action space and physics remain the same.
    """
    metadata = {
        "render_modes": ["human", "rgb_array"],
        "name": "multi_lander_v0",
        "is_parallelizable": True,
        "render_fps": FPS,
    }

    def __init__(
            self,
            render_mode: Optional[str] = None,
            continuous: bool = False,
            gravity: float = -10.0,
            enable_wind: bool = False,
            wind_power: float = 15.0,
            turbulence_power: float = 1.5,
            num_landers: int = 2,
            *args, **kwargs
    ):
        EzPickle.__init__(
            self,
            render_mode,
            continuous,
            gravity,
            enable_wind,
            wind_power,
            turbulence_power,
            *args, **kwargs
        )
        AECEnv.__init__(self)
        self.render_mode = render_mode
        self.np_random = np.random

        # Value Checkers
        assert (-12.0 < gravity and gravity < 0.0
            ), f"gravity (current value: {gravity}) must be between -12 and 0"
        self.gravity = gravity

```

```python
        if 0.0 > wind_power or wind_power > 20.0:
            logger.warn(
                "wind_power value is recommended to be between 0.0 and 20.0, " +
                f"(current value: {wind_power})"
            )
        self.wind_power = wind_power

        if 0.0 > turbulence_power or turbulence_power > 2.0:
            logger.warn(
                "turbulence_power value is recommended to be between 0.0 and " +
                f"2.0, (current value: {turbulence_power})"
            )
        # These are bounds for position realistically the environment
        # should have ended long before we reach more than 50% outside
        low = np.array(
            [
                -2.5, -2.5,              # x,y coordinates
                -10.0, -10.0,            # x,y velocity bounds is 5x rated speed
                -2 * math.pi, -10.0,     # Angle, Angular Velocity
                -0.0, -0.0,              # L,R Leg not on ground
            ]).astype(np.float32)
        high = np.array(
            [
                2.5, 5.0, # 2.5,         # x,y coordinates
                10.0, 10.0,              # x,y velocity bounds is 5x rated speed
                2 * math.pi, 10.0,       # Angle, Angular Velocity
                1.0, 1.0,                # L,R Leg on ground
            ]).astype(np.float32)

        # Environmental Variables
        self.turbulence_power = turbulence_power
        self.enable_wind = enable_wind
        self.screen: pygame.Surface = None
        self.clock = None
        self.isopen = True
        self.world = Box2D.b2World(gravity=(0, gravity))
        self.moon = None
        self.particles = []
        self.landers = []
        self.legs = []
        self.prev_reward = None
        self.continuous = continuous

        # Agents
        self.possible_agents = ["lander_" + str(i) for i in range(num_landers)]
        self.agents = self.possible_agents[:]
        self.agent_name_mapping = dict(zip(self.agents,
                                           list(range(self.num_agents))))
        self._agent_selector = agent_selector(self.agents)

        # Spaces
        self.observation_spaces = dict(zip(self.agents,
                                    [spaces.Box(low, high)]*self.num_agents))
        if self.continuous:
            # Action is two floats [main engine, left-right engines].
            # Main engine: -1..0 off, 0..+1 throttle from 50% to 100% power.
            #              Engine can't work with less than 50% power.
            # Left-right:  -1.0..-0.5 fire left engine, +0.5..+1.0 fire right
            #              engine, -0.5..0.5 off
            self._action_space = spaces.Box(-1, +1, (2,), dtype=np.float32)
```

```python
        else:
            # No-op, fire left engine, main engine, right engine
            self._action_space = spaces.Discrete(4)
        self.action_spaces = dict(zip(self.agents,
                                      [self._action_space]*self.num_agents))

    def observation_space(self, agent):
        return self.observation_spaces[agent]

    def action_space(self, agent):
        return self.action_spaces[agent]

    def convert_to_dict(self, list_of_list):
        return dict(zip(self.agents, list_of_list))

    def close(self):
        if self.screen is not None:
            import pygame
            pygame.display.quit()
            pygame.quit()
            pygame.QUIT
            self.isopen = False

    def create_landers(self):
        landers = []
        for n in range(self.num_agents):
            lander: Box2D.b2Body = self.world.CreateDynamicBody(
                position=((n+1)/(self.num_agents+1) * VIEWPORT_W / SCALE,
                          VIEWPORT_H / SCALE),
                angle=0.0,
                fixtures=fixtureDef(
                    shape=polygonShape(
                        vertices=[(x/SCALE, y/SCALE) for x, y in LANDER_POLY]),
                    density=5.0,
                    friction=0.1,
                    categoryBits=0x0010,
                    #maskBits=0x001,  # uncomment for : collide only with ground
                    restitution=0.0,
                ),  # 0.99 bouncy
            )
            lander.color1 = ((128), (102+75*(n))%255, (230))
            lander.color2 = ((77), (77), (128))
            landers.append(lander)
        self.landers = landers

        legs = []
        for n,lander in enumerate(self.landers):
            for i in [-1, +1]:
                leg = self.world.CreateDynamicBody(
                    position=(lander.position[0] - i * LEG_AWAY / SCALE,
                              lander.position[1]),
                    angle=(i * 0.05),
                    fixtures=fixtureDef(
                        shape=polygonShape(box=(LEG_W / SCALE, LEG_H / SCALE)),
                        density=1.0,
                        restitution=0.0,
                        categoryBits=0x0020,
                        #maskBits=0x001,
                    ),
                )
```

```
                leg.ground_contact = False
                leg.color1 = ((128), (102+75*(n))%255, (230))
                leg.color2 = ((77), (77), (128))
                rjd = revoluteJointDef(
                    bodyA=lander,
                    bodyB=leg,
                    localAnchorA=(0, 0),
                    localAnchorB=(i * LEG_AWAY / SCALE, LEG_DOWN / SCALE),
                    enableMotor=True,
                    enableLimit=True,
                    maxMotorTorque=LEG_SPRING_TORQUE,
                    motorSpeed=+0.3 * i,  # low enough not to jump into the sky
                )
                if i == -1:
                    rjd.lowerAngle = (
                        +0.9 - 0.5
                    )  # Valid angle of travel for the legs
                    rjd.upperAngle = +0.9
                else:
                    rjd.lowerAngle = -0.9
                    rjd.upperAngle = -0.9 + 0.5
                leg.joint = self.world.CreateJoint(rjd)
                legs.append(leg)
        self.legs = legs

    def _create_particle(self, mass, x, y, ttl):
        p = self.world.CreateDynamicBody(
            position=(x, y),
            angle=0.0,
            fixtures=fixtureDef(
                shape=circleShape(radius=2 / SCALE, pos=(0, 0)),
                density=mass,
                friction=0.1,
                categoryBits=0x0100,
                maskBits=0x001,  # collide only with ground
                restitution=0.3,
            ),
        )
        p.ttl = ttl
        self.particles.append(p)
        self._clean_particles(False)
        return p

    def _clean_particles(self, all_particle):
        while self.particles and (all_particle or self.particles[0].ttl < 0):
            self.world.DestroyBody(self.particles.pop(0))


    def render(self):
        if self.render_mode is None:
            assert self.spec is not None
            gym.logger.warn(
                "You are calling render method without specifying render mode."
                "You can specify the render_mode at initialization, "
                f'e.g. gym.make("{self.spec.id}", render_mode="rgb_array")')
            return

        try:
            import pygame
            from pygame import gfxdraw
```

```
        except ImportError as e:
            raise DependencyNotInstalled(
                'pygame is not installed, run `pip install "gymnasium[box2d]"`'
            ) from e

        if self.screen is None and self.render_mode == "human":
            pygame.init()
            pygame.display.init()
            self.screen = pygame.display.set_mode((VIEWPORT_W, VIEWPORT_H))
        if self.clock is None:
            self.clock = pygame.time.Clock()

        self.surf = pygame.Surface((VIEWPORT_W, VIEWPORT_H))
        pygame.transform.scale(self.surf, (SCALE, SCALE))
        pygame.draw.rect(self.surf, (255, 255, 255), self.surf.get_rect())

        for obj in self.particles:
            obj.ttl -= 0.15
            obj.color1 = (
                int(max(0.2, 0.15 + obj.ttl) * 255),
                int(max(0.2, 0.5 * obj.ttl) * 255),
                int(max(0.2, 0.5 * obj.ttl) * 255),
            )
            obj.color2 = (
                int(max(0.2, 0.15 + obj.ttl) * 255),
                int(max(0.2, 0.5 * obj.ttl) * 255),
                int(max(0.2, 0.5 * obj.ttl) * 255),
            )

        self._clean_particles(False)

        for p in self.sky_polys:
            scaled_poly = []
            for coord in p:
                scaled_poly.append((coord[0] * SCALE, coord[1] * SCALE))
            pygame.draw.polygon(self.surf, (0, 0, 0), scaled_poly)
            gfxdraw.aapolygon(self.surf, scaled_poly, (0, 0, 0))

        for obj in self.particles + self.drawlist:
            for f in obj.fixtures:
                trans = f.body.transform
                if type(f.shape) is circleShape:
                    pygame.draw.circle(
                        self.surf,
                        color=obj.color1,
                        center=trans * f.shape.pos * SCALE,
                        radius=f.shape.radius * SCALE,
                    )
                    pygame.draw.circle(
                        self.surf,
                        color=obj.color2,
                        center=trans * f.shape.pos * SCALE,
                        radius=f.shape.radius * SCALE,
                    )
                else:
                    path = [trans * v * SCALE for v in f.shape.vertices]
                    pygame.draw.polygon(self.surf,color=obj.color1,points=path)
                    gfxdraw.aapolygon(self.surf, path, obj.color1)
                    pygame.draw.aalines(
                        self.surf, color=obj.color2, points=path, closed=True)
```

```python
            for x in [self.helipad_x1, self.helipad_x2]:
                x = x * SCALE
                flagy1 = self.helipad_y * SCALE
                flagy2 = flagy1 + 50
                pygame.draw.line(
                    self.surf,
                    color=(255, 255, 255),
                    start_pos=(x, flagy1),
                    end_pos=(x, flagy2),
                    width=1,
                )
                pygame.draw.polygon(
                    self.surf,
                    color=(204, 204, 0),
                    points=[
                        (x, flagy2),
                        (x, flagy2 - 10),
                        (x + 25, flagy2 - 5),
                    ],
                )
                gfxdraw.aapolygon(
                    self.surf,
                    [(x, flagy2), (x, flagy2 - 10), (x + 25, flagy2 - 5)],
                    (204, 204, 0),
                )

        self.surf = pygame.transform.flip(self.surf, False, True)

        if self.render_mode == "human":
            assert self.screen is not None
            self.screen.blit(self.surf, (0, 0))
            pygame.event.pump()
            self.clock.tick(self.metadata["render_fps"])
            pygame.display.flip()
        elif self.render_mode == "rgb_array":
            return np.transpose(
                np.array(pygame.surfarray.pixels3d(self.surf)), axes=(1, 0, 2)
            )
    # End Render()

    def _destroy(self):
        if not self.moon:
            return
        self.world.contactListener = None
        self._clean_particles(True)
        self.world.DestroyBody(self.moon)
        self.moon = None
        for lander in self.landers:
            self.world.DestroyBody(lander)
            lander = None
        for i in range(len(self.legs)):
            self.world.DestroyBody(self.legs[i])


    def reset(self,*,seed: Optional[int] = None,options: Optional[dict] = None):
        self._destroy()
        # Issue: https://github.com/Farama-Foundation/Gymnasium/issues/728
        # self._destroy() is not enough to clean(reset), workaround is
        # to create a totally new world for self.reset()
```

```python
        self.world = Box2D.b2World(gravity=(0, self.gravity))
        self.world.contactListener_keepref = ContactDetector(self)
        self.world.contactListener = self.world.contactListener_keepref
        self.game_over = False
        self.prev_shaping = [None]*self.num_agents

        W = VIEWPORT_W / SCALE
        H = VIEWPORT_H / SCALE

        # Create Terrain
        CHUNKS = 11
        height = self.np_random.uniform(0, H / 2, size=(CHUNKS + 1,))
        chunk_x = [W / (CHUNKS - 1) * i for i in range(CHUNKS)]
        self.helipad_x1 = chunk_x[CHUNKS // 2 - 1]
        self.helipad_x2 = chunk_x[CHUNKS // 2 + 1]
        self.helipad_y = H / 4
        height[CHUNKS // 2 - 2] = self.helipad_y
        height[CHUNKS // 2 - 1] = self.helipad_y
        height[CHUNKS // 2 + 0] = self.helipad_y
        height[CHUNKS // 2 + 1] = self.helipad_y
        height[CHUNKS // 2 + 2] = self.helipad_y
        smooth_y = [
            0.33 * (height[i - 1] + height[i + 0] + height[i + 1])
            for i in range(CHUNKS)
        ]

        self.moon = self.world.CreateStaticBody(
            shapes=edgeShape(vertices=[(0, 0), (W, 0)])
        )
        self.sky_polys = []
        for i in range(CHUNKS - 1):
            p1 = (chunk_x[i], smooth_y[i])
            p2 = (chunk_x[i + 1], smooth_y[i + 1])
            self.moon.CreateEdgeFixture(vertices=[p1,p2],density=0,friction=0.1)
            self.sky_polys.append([p1, p2, (p2[0], H), (p1[0], H)])

        self.moon.color1 = (0.0, 0.0, 0.0)
        self.moon.color2 = (0.0, 0.0, 0.0)

        if self.enable_wind:  # Initialize wind pattern based on index
            self.wind_idx = self.np_random.integers(-9999, 9999)
            self.torque_idx = self.np_random.integers(-9999, 9999)

        self.create_landers()

        for lander in self.landers:
            lander.ApplyForceToCenter(
            (
                self.np_random.uniform(-INITIAL_RANDOM, INITIAL_RANDOM),
                self.np_random.uniform(-INITIAL_RANDOM, INITIAL_RANDOM),
            ),
            True,
        )
        self.drawlist = self.legs + self.landers
        if self.render_mode == "human": self.render()

        self.agents = self.possible_agents[:]
        self._agent_selector = agent_selector(self.agents)
        self.agent_selection = self._agent_selector.next()
        self.rewards = {agent: 0 for agent in self.agents}
```

```
        self._cumulative_rewards = {agent: 0 for agent in self.agents}
        self.terminations = {agent: False for agent in self.agents}
        self.truncations = {agent: False for agent in self.agents}
        self.infos = {agent: {} for agent in self.agents}

        return self.observe(), None
    # End reset()


    def observe(self, agent=None):
        if not agent : agent = self.agent_selection
        id = self.agent_name_mapping[agent]
        lander = self.landers[id]
        pos = lander.position
        vel = lander.linearVelocity

        observation = [
            (pos.x - VIEWPORT_W/SCALE/2) / (VIEWPORT_W/SCALE/2),
            (pos.y - (self.helipad_y + LEG_DOWN/SCALE)) / (VIEWPORT_H/SCALE/2),
            vel.x * (VIEWPORT_W/SCALE/2) / FPS,
            vel.y * (VIEWPORT_H/SCALE/2) / FPS,
            lander.angle,
            20.0 * lander.angularVelocity / FPS,
            1.0 if self.legs[id*2+0].ground_contact else 0.0,
            1.0 if self.legs[id*2+1].ground_contact else 0.0,
        ]
        assert len(observation) == 8
        return observation

    def latest_reward_state(self, agent, state):
        reward = 0
        id = self.agent_name_mapping[agent]
        shaping = (
            -100 * np.sqrt(state[0] * state[0] + state[1] * state[1])
            - 100 * np.sqrt(state[2] * state[2] + state[3] * state[3])
            - 100 * abs(state[4])
            + 10 * state[6] + 10 * state[7]
        )   # And ten points for legs contact, the idea is if you
            # lose contact again after landing, you get negative reward
        if self.prev_shaping[id] is not None:
            reward = shaping - self.prev_shaping[id]
        self.prev_shaping[id] = shaping
        return reward


    def step(self, action):
        assert self.landers is not None, "You forgot to call reset()"
        if (
            self.terminations[self.agent_selection]
            or self.truncations[self.agent_selection]
        ):
            # If one agent has terminated this accepts a None action,
            # which otherwise errors, handles stepping to the next agent
            obs = self.observe(self.agent_selection)
            self.agent_selection = self._agent_selector.next()
            return obs, 0, True, False, {}

        # Update wind and apply to the lander
        if self.enable_wind and not any([l.ground_contact for l in self.legs]):
            # the function used for wind is tanh(sin(2 k x) + sin(pi k x)),
```

```python
            # which is proven to never be periodic, k = 0.01
            wind_mag = (
                math.tanh(
                    math.sin(0.02 * self.wind_idx)
                    + (math.sin(math.pi * 0.01 * self.wind_idx))
                ) * self.wind_power
            )
            self.wind_idx += 1

            for lander in self.landers:
                lander.ApplyForceToCenter(
                    (wind_mag, 0.0),
                    True,
                )

            # the function used for torque is tanh(sin(2 k x) + sin(pi k x)),
            # which is proven to never be periodic, k = 0.01
            torque_mag = (
                math.tanh(
                    math.sin(0.02 * self.torque_idx)
                    + (math.sin(math.pi * 0.01 * self.torque_idx))
                ) * self.turbulence_power
            )
            self.torque_idx += 1

            for lander in self.landers:
                lander.ApplyTorque(
                    torque_mag,
                    True,
                )

        # For current agent:
        agent = self.agent_selection
        lander = self.landers[self.agent_name_mapping[agent]]
        # Check action validity
        if self.continuous:
            action = np.clip(action, -1, +1).astype(np.float64)
        else:
            assert self.action_spaces[agent].contains(
                action), f"{action!r} ({type(action)}) invalid "

        # Tip is the (X and Y) components of the rotation of the lander.
        tip = (math.sin(lander.angle), math.cos(lander.angle))

        # Side is the (-Y and X) components of the rotation of the lander.
        side = (-tip[1], tip[0])

        # Generate two random numbers between -1/SCALE and 1/SCALE.
        dispersion = [self.np_random.uniform(-1.0,+1.0)/SCALE for _ in range(2)]

        m_power = 0.0
        if (self.continuous and action[0] > 0.0) or (
            not self.continuous and action == 2
        ):
            # Main engine
            if self.continuous:
                m_power = (np.clip(action[0], 0.0, 1.0) + 1.0) * 0.5  # 0.5..1.0
                assert m_power >= 0.5 and m_power <= 1.0
            else:
                m_power = 1.0
```

```
            # 4 is move a bit downwards, +-2 for randomness
            # The components of the impulse to be applied by the main engine.
            ox = (
                tip[0] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
                + side[0] * dispersion[1]
            )
            oy = (
                -tip[1] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
                - side[1] * dispersion[1]
            )
            impulse_pos = (lander.position[0] + ox, lander.position[1] + oy)
            if self.render_mode is not None:
                # particles are just a decoration, with no physics impact,
                # so don't add them when not rendering
                p = self._create_particle(
                    3.5,  # 3.5 is here to make particle speed adequate
                    impulse_pos[0],
                    impulse_pos[1],
                    m_power,
                )
                p.ApplyLinearImpulse(
                    (
                        ox * MAIN_ENGINE_POWER * m_power,
                        oy * MAIN_ENGINE_POWER * m_power,
                    ),
                    impulse_pos,
                    True,
                )
            lander.ApplyLinearImpulse(
                (-ox * MAIN_ENGINE_POWER * m_power,
                 -oy * MAIN_ENGINE_POWER * m_power),
                impulse_pos,
                True,
            )

        s_power = 0.0
        if (self.continuous and np.abs(action[1]) > 0.5) or (
            not self.continuous and action in [1, 3]
        ):
            # Orientation/Side engines
            if self.continuous:
                direction = np.sign(action[1])
                s_power = np.clip(np.abs(action[1]), 0.5, 1.0)
                assert s_power >= 0.5 and s_power <= 1.0
            else:
                # action = 1 is left, action = 3 is right
                direction = action - 2
                s_power = 1.0

            # The components of the impulse to be applied by the side engines.
            ox = tip[0] * dispersion[0] + side[0] * (
                3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
            )
            oy = -tip[1] * dispersion[0] - side[1] * (
                3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
            )

            # The constant 17 is presumably meant to be SIDE_ENGINE_HEIGHT.
            # However, SIDE_ENGINE_HEIGHT is defined as 14, causing the
```

```python
            # position of the thrust on the body of the lander to change,
            # depending on the orientation of the lander. This results in
            # an orientation dependent torque being applied to the lander.

            impulse_pos = (
                lander.position[0] + ox - tip[0] * 17 / SCALE,
                lander.position[1] + oy + tip[1] * SIDE_ENGINE_HEIGHT / SCALE,
            )
            if self.render_mode is not None:
                # particles are just decoration, with no impact on the physics,
                # so don't add them when not rendering
                p = self._create_particle(0.7, impulse_pos[0],
                                          impulse_pos[1], s_power)
                p.ApplyLinearImpulse(
                    (ox * SIDE_ENGINE_POWER * s_power,
                     oy * SIDE_ENGINE_POWER * s_power,),
                    impulse_pos,
                    True,
                )
            lander.ApplyLinearImpulse(
                (-ox * SIDE_ENGINE_POWER * s_power,
                 -oy * SIDE_ENGINE_POWER * s_power),
                impulse_pos,
                True,
            )

        # Update Positions
        self.world.Step(1.0 / FPS, 6 * 30, 2 * 30)

        observation = self.observe(agent)
        reward = self.latest_reward_state(agent, observation)
        reward -= m_power * 0.30 # Deduct cost of fuel
        reward -= s_power * 0.03 # Deduct cost of fuel

        if self.game_over or abs(observation[0]) >= 1.0:
            self.terminations[self.agent_selection] = True
            reward = -100
        if not lander.awake:
            self.terminations[self.agent_selection] = True
            reward = +100
        terminated = self.terminations[self.agent_selection]

        self.rewards[agent] = reward
        self._accumulate_rewards()

        self.agent_selection = self._agent_selector.next()

        if self.render_mode == "human":
            self.render()
        # truncation=False as the time limit is handled by the `TimeLimit`
        return observation, reward, terminated, False, {}


class Parallel_Env(SequentialEnv):
    def observe(self):
        obs_list = []
        for i,lander in enumerate(self.landers):
            pos = lander.position
            vel = lander.linearVelocity
            obs = [
```

```python
                (pos.x - VIEWPORT_W/SCALE/2) / (VIEWPORT_W/SCALE/2),
                (pos.y - (self.helipad_y + LEG_DOWN/SCALE)) / (VIEWPORT_H/SCALE/2),
                vel.x * (VIEWPORT_W/SCALE/2) / FPS,
                vel.y * (VIEWPORT_H/SCALE/2) / FPS,
                lander.angle,
                20.0 * lander.angularVelocity / FPS,
                1.0 if self.legs[i*2+0].ground_contact else 0.0,
                1.0 if self.legs[i*2+1].ground_contact else 0.0,
            ]
            assert len(obs) == 8
            obs_list.append(obs)
        return dict(zip(self.possible_agents,obs_list))

    def last(self):
        return self.observe()

    def step(self, action_list):
        self._clear_rewards()
        assert self.landers is not None, "You forgot to call reset()"

        # Update wind and apply to the lander
        if self.enable_wind and not any([l.ground_contact for l in self.legs]):
            # the function used for wind is tanh(sin(2 k x) + sin(pi k x)),
            # which is proven to never be periodic, k = 0.01
            wind_mag = (
                math.tanh(
                    math.sin(0.02 * self.wind_idx)
                    + (math.sin(math.pi * 0.01 * self.wind_idx))
                ) * self.wind_power
            )
            self.wind_idx += 1

            for lander in self.landers:
                lander.ApplyForceToCenter(
                    (wind_mag, 0.0),
                    True,
                )
            # the function used for torque is tanh(sin(2 k x) + sin(pi k x)),
            # which is proven to never be periodic, k = 0.01
            torque_mag = (
                math.tanh(
                    math.sin(0.02 * self.torque_idx)
                    + (math.sin(math.pi * 0.01 * self.torque_idx))
                ) * self.turbulence_power
            )
            self.torque_idx += 1

            for lander in self.landers:
                lander.ApplyTorque(
                    torque_mag,
                    True,
                )

        #for agent in self.agents:
        for agent,action in action_list.items():
            # For current agent:
            lander = self.landers[self.agent_name_mapping[agent]]
            # Check action validity
            if self.continuous:
                action = np.clip(action, -1, +1).astype(np.float64)
```

```
            else:
                assert self.action_spaces[agent].contains(
                    action), f"{action!r} ({type(action)}) invalid "

            # Tip is the (X and Y) components of the rotation of the lander.
            tip = (math.sin(lander.angle), math.cos(lander.angle))

            # Side is the (-Y and X) components of the rotation of the lander.
            side = (-tip[1], tip[0])

            # Generate two random numbers between -1/SCALE and 1/SCALE.
            dispersion = [self.np_random.uniform(-1.0,+1.0)/SCALE for _ in range(2)]

            m_power = 0.0
            if (self.continuous and action[0] > 0.0) or (
                not self.continuous and action == 2
            ):
                # Main engine
                if self.continuous:
                    m_power = (np.clip(action[0], 0.0, 1.0) + 1.0) * 0.5  # 0.5..1.0
                    assert m_power >= 0.5 and m_power <= 1.0
                else:
                    m_power = 1.0

                # 4 is move a bit downwards, +-2 for randomness
                # The components of the impulse to be applied by the main engine.
                ox = (
                    tip[0] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
                    + side[0] * dispersion[1]
                )
                oy = (
                    -tip[1] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
                    - side[1] * dispersion[1]
                )
                impulse_pos = (lander.position[0] + ox, lander.position[1] + oy)
                if self.render_mode is not None:
                    # particles are just a decoration, with no physics impact,
                    # so don't add them when not rendering
                    p = self._create_particle(
                        3.5,  # 3.5 is here to make particle speed adequate
                        impulse_pos[0],
                        impulse_pos[1],
                        m_power,
                    )
                    p.ApplyLinearImpulse(
                        (
                            ox * MAIN_ENGINE_POWER * m_power,
                            oy * MAIN_ENGINE_POWER * m_power,
                        ),
                        impulse_pos,
                        True,
                    )
                lander.ApplyLinearImpulse(
                    (-ox * MAIN_ENGINE_POWER * m_power,
                    -oy * MAIN_ENGINE_POWER * m_power),
                    impulse_pos,
                    True,
                )

            s_power = 0.0
```

```
    if (self.continuous and np.abs(action[1]) > 0.5) or (
        not self.continuous and action in [1, 3]
    ):
        # Orientation/Side engines
        if self.continuous:
            direction = np.sign(action[1])
            s_power = np.clip(np.abs(action[1]), 0.5, 1.0)
            assert s_power >= 0.5 and s_power <= 1.0
        else:
            # action = 1 is left, action = 3 is right
            direction = action - 2
            s_power = 1.0

        # The components of the impulse to be applied by the side engines.
        ox = tip[0] * dispersion[0] + side[0] * (
            3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
        )
        oy = -tip[1] * dispersion[0] - side[1] * (
            3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
        )

        # The constant 17 is presumably meant to be SIDE_ENGINE_HEIGHT.
        # However, SIDE_ENGINE_HEIGHT is defined as 14, causing the
        # position of the thrust on the body of the lander to change,
        # depending on the orientation of the lander. This results in
        # an orientation dependent torque being applied to the lander.

        impulse_pos = (
            lander.position[0] + ox - tip[0] * 17 / SCALE,
            lander.position[1] + oy + tip[1] * SIDE_ENGINE_HEIGHT / SCALE,
        )
        if self.render_mode is not None:
            # particles are just decoration, with no impact on the physics,
            # so don't add them when not rendering
            p = self._create_particle(0.7, impulse_pos[0],
                                    impulse_pos[1], s_power)
            p.ApplyLinearImpulse(
                (ox * SIDE_ENGINE_POWER * s_power,
                oy * SIDE_ENGINE_POWER * s_power,),
                impulse_pos,
                True,
            )
        lander.ApplyLinearImpulse(
            (-ox * SIDE_ENGINE_POWER * s_power,
            -oy * SIDE_ENGINE_POWER * s_power),
            impulse_pos,
            True,
        )
        self.rewards[agent] -= m_power * 0.30 # Deduct cost of fuel
        self.rewards[agent] -= s_power * 0.03 # Deduct cost of fuel
""" End Agent Loop """

# Update Positions
self.world.Step(1.0 / FPS, 6 * 30, 2 * 30)

obs_list = self.observe()

for agent in self.agents:
    obs = obs_list[agent]
    lander = self.landers[self.agent_name_mapping[agent]]
```

```python
                self.rewards[agent] += self.latest_reward_state(agent, obs)
                if self.game_over or abs(obs[0]) >= 1.0:
                    self.terminations[agent] = True
                    self.rewards[agent] -= 100
                if not lander.awake:
                    self.terminations[agent] = True
                    self.rewards[agent] += 100

            self._accumulate_rewards()

            if self.render_mode == "human":
                self.render()
            # truncation=False as the time limit is handled by the `TimeLimit`
            return obs_list, self.rewards, self.terminations, self.truncations, {}


def heuristic(env, s):
    """
    The heuristic for
    1. Testing
    2. Demonstration rollout.

    Args:
        env: The environment
        s (list): The state. Attributes:
            s[0] is the horizontal coordinate
            s[1] is the vertical coordinate
            s[2] is the horizontal speed
            s[3] is the vertical speed
            s[4] is the angle
            s[5] is the angular speed
            s[6] 1 if first leg has contact, else 0
            s[7] 1 if second leg has contact, else 0

    Returns:
         a: The heuristic to be fed into the step function defined above to
            determine the next step and reward.
    """

    angle_targ = s[0] * 0.5 + s[2] * 1.0  # angle should point towards center
    if angle_targ > 0.4:
        angle_targ = 0.4  # more than 0.4 radians (22 degrees) is bad
    if angle_targ < -0.4:
        angle_targ = -0.4
    hover_targ = 0.55 * np.abs(
        s[0]
    )  # target y should be proportional to horizontal offset

    angle_todo = (angle_targ - s[4]) * 0.5 - (s[5]) * 1.0
    hover_todo = (hover_targ - s[1]) * 0.5 - (s[3]) * 0.5

    if s[6] or s[7]:  # legs have contact
        angle_todo = 0
        hover_todo = (
            -(s[3]) * 0.5
        )  # override to reduce fall speed, that's all we need after contact

    if env.unwrapped.continuous:
        a = np.array([hover_todo * 20 - 1, -angle_todo * 20])
        a = np.clip(a, -1, +1)
```

```
    else:
        a = 0
        if hover_todo > np.abs(angle_todo) and hover_todo > 0.05:
            a = 2
        elif angle_todo < -0.05:
            a = 3
        elif angle_todo > +0.05:
            a = 1
    return a

def demo_heuristic_lander(env, reps=1, seed=None, render=False):
    total_reward = 0
    steps = 0
    for _ in range(reps):
        s = env.reset(seed=seed)
        while True:
            s = env.last()[0]
            a = heuristic(env, s)
            s, r, terminated, truncated, info = env.step(a)
            total_reward += r

            if render:
                still_open = env.render()
                if still_open is False:
                    break

            if steps % 20 == 0 or terminated or truncated:
                print("observations:", " ".join([f"{x:+0.2f}" for x in s]))
                print(f"step {steps} total_reward {total_reward:+0.2f}")
                print(env.terminations.values())
            steps += 1
            # if terminated or truncated:
            if all(env.terminations.values()):
                break
    if render:
        env.close()
    return total_reward

def demo_parallel_heuristic(env, reps=1, seed=None, render=False):
    total_reward = 0
    steps = 0
    for _ in range(reps):
        _ = env.reset(seed=seed)
        obs = env.last()
        while True:
            actions = {agent: heuristic(env, obs[agent]) for agent in env.agents}
            obs, r, terminateds, truncateds, info = env.step(actions)
            total_reward += sum(r.values())

            if render:
                still_open = env.render()
                if still_open is False:
                    break

            if steps % 20 == 0 or all(terminateds) or all(truncateds):
                for agent in env.agents:
                    print(f"{agent} observations:", " ".join([f"{x:+0.2f}" for x in obs[ag
                print(f"step {steps} total_reward {total_reward:+0.2f}")
                print(env.terminations.values())
            steps += 1
```

```python
            # if terminated or truncated:
            if all(env.terminations.values()):
                break
    if render:
        env.close()
    return total_reward

def parallel_env(**kwargs):
    return Parallel_Env(**kwargs)

def sequential_env(**kwargs):
    return SequentialEnv(**kwargs)

def env(**kwargs):
    return Parallel_Env(**kwargs)

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description=
                                     'A multi-agent version of Lunar Lander')
    parser.add_argument('-s', '--sequential', action="store_true",
                        help='Iterate as AEC environment')
    parser.add_argument('-n', '--num_landers', type=int, default=2,
                        help='number of landers')
    parser.add_argument('-d', '--demo_iters', type=int, default=1,
                        help='number of landers')
    args = parser.parse_args()

    if args.sequential:
        _env = sequential_env(render_mode="human", num_landers=args.num_landers)
        demo_heuristic_lander(_env, reps=args.demo_iters, render=True)
    else:
        _env = parallel_env(render_mode="human", num_landers=args.num_landers)
        demo_parallel_heuristic(_env, reps=args.demo_iters, render=True)
```

C ANOVA Tables

C.1 Baseline Lander

```
OLS Regression Results
==============================================================================
Dep. Variable:                  Score   R-squared:                       0.966
Model:                            OLS   Adj. R-squared:                  0.902
Method:                 Least Squares   F-statistic:                     15.12
Date:                Mon, 22 Jul 2024   Prob (F-statistic):           5.46e-05
Time:                        22:33:51   Log-Likelihood:                -131.47
No. Observations:                  30   AIC:                             302.9
Df Residuals:                      10   BIC:                             331.0
Df Model:                          19
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
Intercept               70.1409    135.922      0.516      0.617    -232.712     372.994
tau                  -5.201e+07   2.25e+07     -2.313      0.043   -1.02e+08    -1.9e+06
sigma                   33.4494     24.068      1.390      0.195     -20.178      87.077
n_step                 844.5087    373.852      2.259      0.047      11.515    1677.502
adam_eps               891.1404    278.946      3.195      0.010     269.611    1512.670
ep_len_mean             -0.0763      0.129     -0.589      0.569      -0.365       0.212
tau_pow_2             1020.8171    418.234      2.441      0.035      88.934    1952.700
tau_sigma             2.216e+06   1.12e+06      1.981      0.076   -2.76e+05    4.71e+06
tau_n_step            1.156e+07    7.5e+06      1.542      0.154   -5.15e+06    2.83e+07
tau_adam_eps            4.7e+07   2.06e+07      2.283      0.046    1.12e+06    9.29e+07
tau_ep_len_mean       3824.1099   8513.828      0.449      0.663   -1.51e+04    2.28e+04
sigma_pow_2              5.0039      1.093      4.580      0.001       2.569       7.438
sigma_n_step          -158.6486     39.804     -3.986      0.003    -247.337     -69.960
sigma_adam_eps         -66.9395     34.419     -1.945      0.080    -143.629       9.750
sigma_ep_len_mean       -0.0181      0.008     -2.187      0.054      -0.037       0.000
n_step_pow_2           -37.5373    228.941     -0.164      0.873    -547.650     472.576
n_step_adam_eps      -1089.7884    257.119     -4.238      0.002   -1662.686    -516.891
n_step_ep_len_mean      -0.0624      0.116     -0.539      0.602      -0.320       0.195
adam_eps_pow_2        -123.4370    238.454     -0.518      0.616    -654.745     407.871
adam_eps_ep_len_mean    -0.0525      0.071     -0.740      0.476      -0.211       0.106
ep_len_mean_pow_2     3.182e-05   2.77e-05      1.150      0.277   -2.98e-05    9.35e-05
==============================================================================
Omnibus:                        5.720   Durbin-Watson:                   1.321
Prob(Omnibus):                  0.057   Jarque-Bera (JB):                4.528
Skew:                          -0.944   Prob(JB):                        0.104
Kurtosis:                       3.246   Cond. No.                     6.21e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 8.67e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

C.2 Single-Agent Control

```
OLS Regression Results
==============================================================================
Dep. Variable:                  Score   R-squared:                       0.868
Model:                            OLS   Adj. R-squared:                  0.616
Method:                 Least Squares   F-statistic:                     3.449
Date:                Tue, 23 Jul 2024   Prob (F-statistic):             0.0247
Time:                        22:13:46   Log-Likelihood:                -123.07
No. Observations:                  30   AIC:                             286.1
Df Residuals:                      10   BIC:                             314.2
Df Model:                          19
Covariance Type:            nonrobust
========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
Intercept               415.8797    180.210      2.308      0.044      14.346     817.413
tau                    1.001e+04   1.98e+06      0.005      0.996   -4.39e+06    4.41e+06
sigma                   -62.7730     18.266     -3.437      0.006    -103.473     -22.073
n_step                 -168.8124    259.877     -0.650      0.531    -747.856     410.231
adam_eps                 87.5099    200.547      0.436      0.672    -359.336     534.356
ep_len_mean               0.1006      0.244      0.413      0.688      -0.442       0.643
tau_pow_2               311.4753    314.546      0.990      0.345    -389.377    1012.328
tau_sigma             -2.468e+04   2.05e+05     -0.121      0.906   -4.81e+05    4.31e+05
tau_n_step             9.437e+06   6.03e+06      1.566      0.148   -3.99e+06    2.29e+07
tau_adam_eps          -4.769e+06   3.13e+06     -1.522      0.159   -1.18e+07    2.21e+06
tau_ep_len_mean       -2349.7876   2023.141     -1.161      0.272   -6857.627    2158.052
sigma_pow_2               2.4038      0.599      4.012      0.002       1.069       3.739
sigma_n_step              1.2135     10.117      0.120      0.907     -21.329      23.756
sigma_adam_eps            8.5121      5.914      1.439      0.181      -4.665      21.689
sigma_ep_len_mean         0.0263      0.010      2.674      0.023       0.004       0.048
n_step_pow_2             53.2984    129.963      0.410      0.690    -236.278     342.875
n_step_adam_eps         231.7169    252.934      0.916      0.381    -331.855     795.289
n_step_ep_len_mean       -0.0407      0.226     -0.180      0.861      -0.545       0.464
adam_eps_pow_2         -126.2286    110.253     -1.145      0.279    -371.888     119.430
adam_eps_ep_len_mean     -0.1330      0.112     -1.188      0.262      -0.383       0.116
ep_len_mean_pow_2      -8.01e-06      0.000     -0.056      0.957      -0.000       0.000
==============================================================================
Omnibus:                        8.754   Durbin-Watson:                   1.550
Prob(Omnibus):                  0.013   Jarque-Bera (JB):                7.490
Skew:                          -0.903   Prob(JB):                       0.0236
Kurtosis:                       4.651   Cond. No.                     5.78e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.72e-19. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

C.3 No Parameter Sharing

```
OLS Regression Results
==============================================================================
Dep. Variable:                  Score   R-squared:                       0.864
Model:                            OLS   Adj. R-squared:                  0.605
Method:                 Least Squares   F-statistic:                     3.339
Date:                Tue, 23 Jul 2024   Prob (F-statistic):             0.0276
Time:                        12:14:05   Log-Likelihood:                -138.94
No. Observations:                  30   AIC:                             317.9
Df Residuals:                      10   BIC:                             345.9
Df Model:                          19
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
Intercept              425.0804    244.891      1.736      0.113    -120.571     970.732
tau                  -2.892e+06   2.95e+06     -0.980      0.350   -9.47e+06    3.68e+06
sigma                  -13.7033     25.615     -0.535      0.604     -70.776      43.370
n_step                 -60.6324    363.894     -0.167      0.871    -871.440     750.175
adam_eps              -101.7131    477.938     -0.213      0.836   -1166.626     963.200
ep_len_mean              0.0414      0.187      0.222      0.829      -0.375       0.458
tau_pow_2              -73.8498    253.375     -0.291      0.777    -638.404     490.704
tau_sigma             -9.58e+04   2.43e+05     -0.394      0.702   -6.37e+05    4.45e+05
tau_n_step           -1.973e+06   4.24e+06     -0.465      0.652   -1.14e+07    7.47e+06
tau_adam_eps          3.361e+06   3.01e+06      1.117      0.290   -3.34e+06    1.01e+07
tau_ep_len_mean       1460.5816    946.981      1.542      0.154    -649.424    3570.587
sigma_pow_2              0.4027      1.422      0.283      0.783      -2.765       3.571
sigma_n_step           -17.9749     30.768     -0.584      0.572     -86.530      50.580
sigma_adam_eps          20.5071     19.299      1.063      0.313     -22.494      63.508
sigma_ep_len_mean        0.0005      0.006      0.087      0.933      -0.012       0.013
n_step_pow_2             5.5271    268.271      0.021      0.984    -592.218     603.273
n_step_adam_eps        100.0307    249.637      0.401      0.697    -456.195     656.256
n_step_ep_len_mean      -0.0024      0.080     -0.030      0.977      -0.180       0.175
adam_eps_pow_2          46.3900    267.727      0.173      0.866    -550.143     642.923
adam_eps_ep_len_mean    -0.0622      0.095     -0.654      0.528      -0.274       0.150
ep_len_mean_pow_2    -2.231e-05   3.99e-05     -0.560      0.588      -0.000    6.65e-05
==============================================================================
Omnibus:                        2.859   Durbin-Watson:                   1.983
Prob(Omnibus):                  0.239   Jarque-Bera (JB):                1.678
Skew:                          -0.548   Prob(JB):                        0.432
Kurtosis:                       3.378   Cond. No.                     1.63e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.28e-18. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

C.4 Full Parameter Sharing

```
OLS Regression Results
==============================================================================
Dep. Variable:                  Score   R-squared:                       0.953
Model:                            OLS   Adj. R-squared:                  0.840
Method:                 Least Squares   F-statistic:                     8.450
Date:                Tue, 23 Jul 2024   Prob (F-statistic):            0.00219
Time:                        12:16:56   Log-Likelihood:                -128.54
No. Observations:                  28   AIC:                             297.1
Df Residuals:                       8   BIC:                             323.7
Df Model:                          19
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
Intercept              175.2457    162.159      1.081      0.311    -198.692     549.184
tau                    186.4124    296.485      0.629      0.547    -497.284     870.109
sigma                 1043.0267    402.067      2.594      0.032     115.859    1970.194
n_step                2.574e+06   9.38e+06      0.274      0.791   -1.91e+07    2.42e+07
adam_eps               -95.1023     52.882     -1.798      0.110    -217.048      26.843
ep_len_mean             -0.0065      0.113     -0.057      0.956      -0.266       0.253
tau_pow_2              190.7520    308.769      0.618      0.554    -521.270     902.774
tau_sigma              287.9223    339.789      0.847      0.421    -495.633    1071.478
tau_n_step            2.957e+07   2.43e+07      1.215      0.259   -2.65e+07    8.57e+07
tau_adam_eps           -11.6668     25.878     -0.451      0.664     -71.341      48.008
tau_ep_len_mean         -0.1573      0.138     -1.140      0.287      -0.476       0.161
sigma_pow_2           -723.8557    329.556     -2.196      0.059   -1483.814      36.103
sigma_n_step          1.668e+07   1.05e+07      1.592      0.150   -7.48e+06    4.08e+07
sigma_adam_eps         -64.5262     24.704     -2.612      0.031    -121.494      -7.559
sigma_ep_len_mean       -0.1967      0.106     -1.850      0.101      -0.442       0.048
n_step_pow_2          6058.0985   5450.747      1.111      0.299   -6511.346    1.86e+04
n_step_adam_eps      -1.209e+06    1.3e+06     -0.933      0.378    -4.2e+06    1.78e+06
n_step_ep_len_mean   -1.043e+04   6264.673     -1.665      0.135   -2.49e+04    4018.744
adam_eps_pow_2           5.8045      2.667      2.176      0.061      -0.346      11.955
adam_eps_ep_len_mean     0.0344      0.019      1.799      0.110      -0.010       0.078
ep_len_mean_pow_2     1.923e-05   3.59e-05      0.535      0.607   -6.36e-05       0.000
==============================================================================
Omnibus:                        1.813   Durbin-Watson:                   2.036
Prob(Omnibus):                  0.404   Jarque-Bera (JB):                1.473
Skew:                          -0.545   Prob(JB):                        0.479
Kurtosis:                       2.725   Cond. No.                     1.20e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.25e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```