

Contents

1 Dr. Yielding's Questions	1
2 Dr. Robbins' Questions	3
Multi-Agent Lunar Lander	5
3 Dr. Cox's Questions	12
Mini-Paper	14
Introduction	14
Related Work	14
Methodology	15
Early Results	16
Future Work	17
References	18
A Experiment Running Script	21
B Multi-Agent Lunar Lander	26
C ANOVA Tables	44
C.1 Baseline Lander	44
C.2 Single-Agent Control	45
C.3 No Parameter Sharing	46
C.4 Full Parameter Sharing	47

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.

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

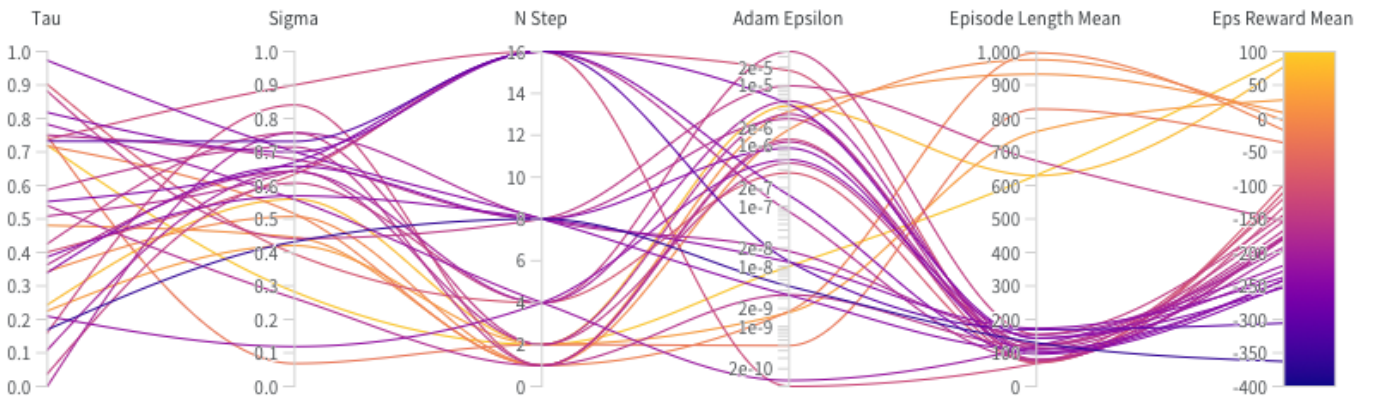


Figure 1: Lunar Lander Baseline Experiment 1 Hyperparameters

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

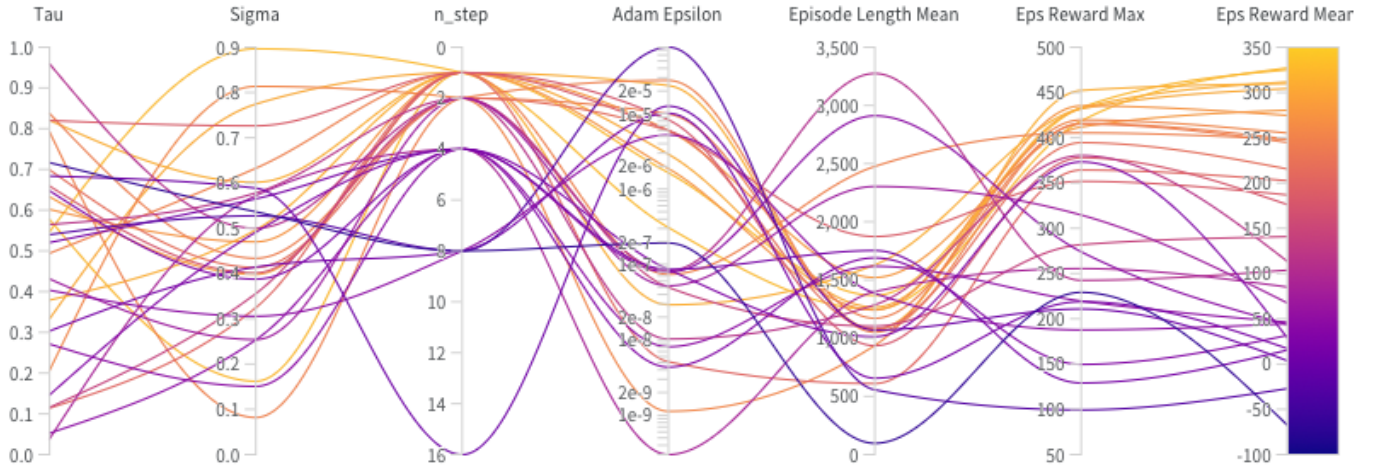


Figure 2: Lunar Lander Baseline Experiment 2 Hyperparameters

Config parameter	Importance ① ↓	Correlation
n_step	<div><div></div></div>	<div><div></div></div>
adam_epsilon	<div><div></div></div>	<div><div></div></div>
sigma0	<div><div></div></div>	<div><div></div></div>
tau	<div><div></div></div>	<div><div></div></div>

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:

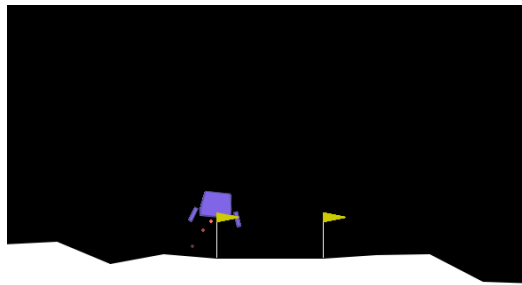
To evaluate the parameters of the sweep I elect to conduct a comparison with three different methods. I will use the same methods for the sweeps performed on the subsequent questions as well.

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.

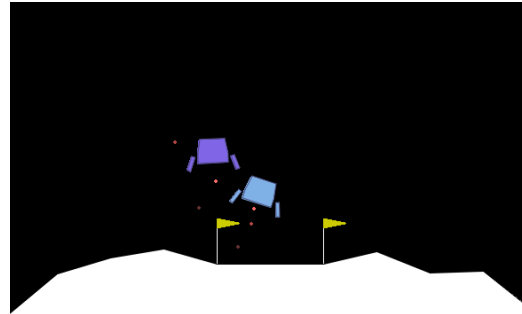
Figure 2 shows the parallel coordinates for this experiment’s parameter sweep. Immediately this suggests that the n-step instances under-perform the single step instances. The factor importance (fig. 3) corroborates this. Finally, we execute the ANOVA in appendix C.1, and find that the σ parameter has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

Multi-Agent Lunar Lander

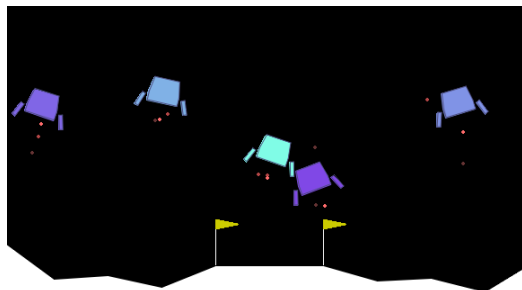
To modify the Lunar Lander environment to support multiple agents it appeared to be possible to simply duplicate all of the sections that referenced the lander in the original code. However, to achieve cleaner code, and behavior more consistent with Farama’s PettingZoo environments, I elected to rewrite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



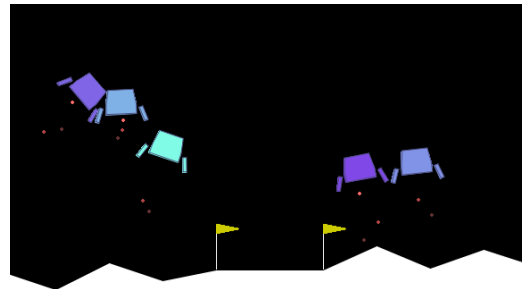
(a) Original



(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

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

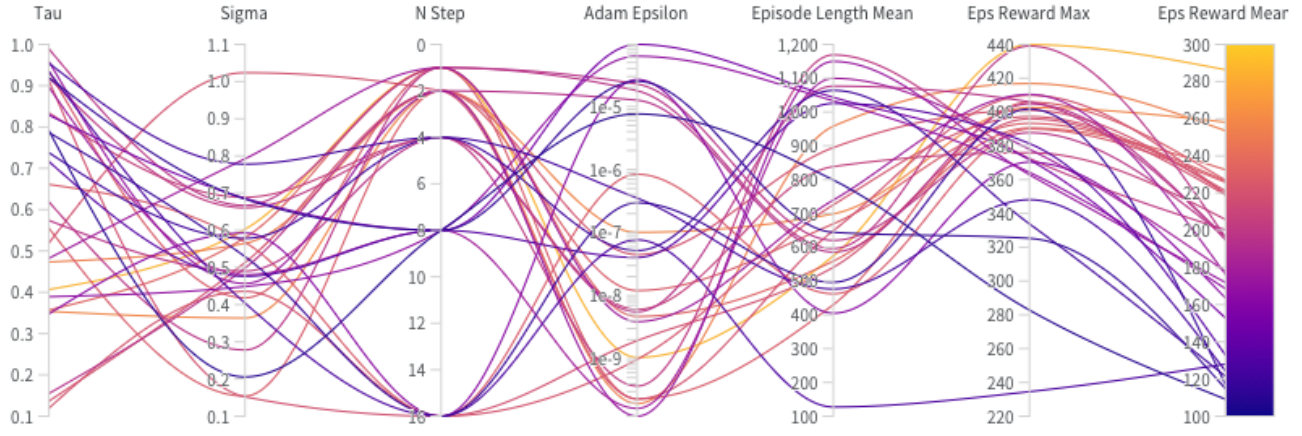


Figure 5: Single-agent Control Hyperparameter Sweep

Config parameter	Importance ① ↓	Correlation
tau	<div><div></div></div>	<div><div></div></div>
n_step	<div><div></div></div>	<div><div></div></div>
sigma0	<div><div></div></div>	<div><div></div></div>
adam_epsilon	<div><div></div></div>	<div><div></div></div>

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 $\pi(b | [s_n]_{n \in N})$.

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.

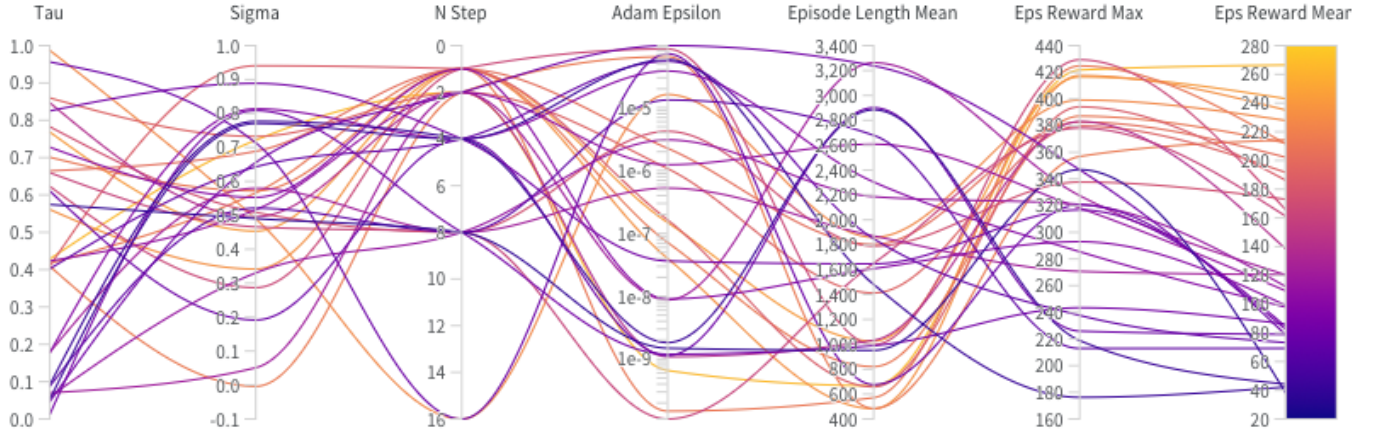


Figure 7: No Parameter Sharing Hyperparameter Sweep

Config parameter	Importance ① ↓	Correlation
tau	<div><div></div></div>	<div><div></div></div>
n_step	<div><div></div></div>	<div><div></div></div>
sigma0	<div><div></div></div>	<div><div></div></div>
adam_epsilon	<div><div></div></div>	<div><div></div></div>

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.

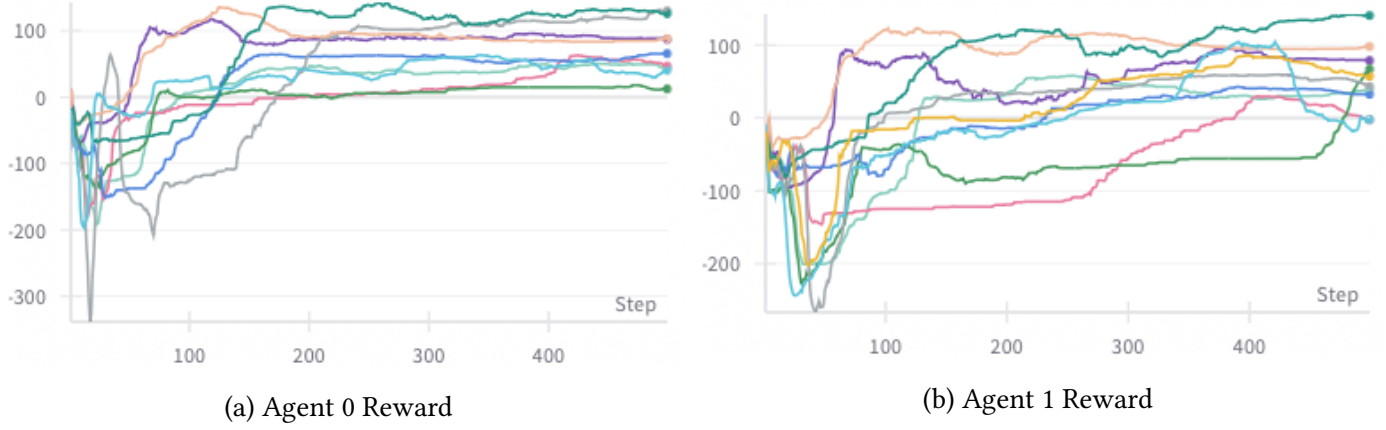


Figure 9: No-parameter Sharing Learning Curves, average reward \times training iteration

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 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
```

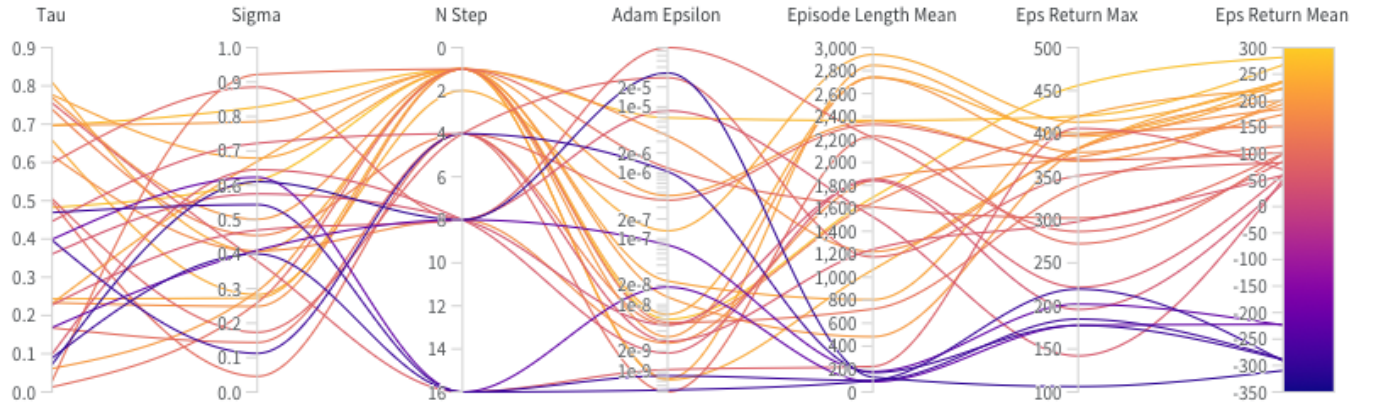


Figure 10: Full Parameter Sharing Hyperparameter Sweep

Config parameter	Importance $\uparrow \downarrow$	Correlation
n_step	<div style="width: 40%; background-color: #000080;"></div>	<div style="width: 60%; background-color: #DC143C;"></div>
tau	<div style="width: 20%; background-color: #000080;"></div>	<div style="width: 80%; background-color: #008000;"></div>
adam_epsilon	<div style="width: 10%; background-color: #000080;"></div>	<div style="width: 90%; background-color: #FF69B4;"></div>
sigma0	<div style="width: 5%; background-color: #000080;"></div>	<div style="width: 95%; background-color: #008000;"></div>

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.

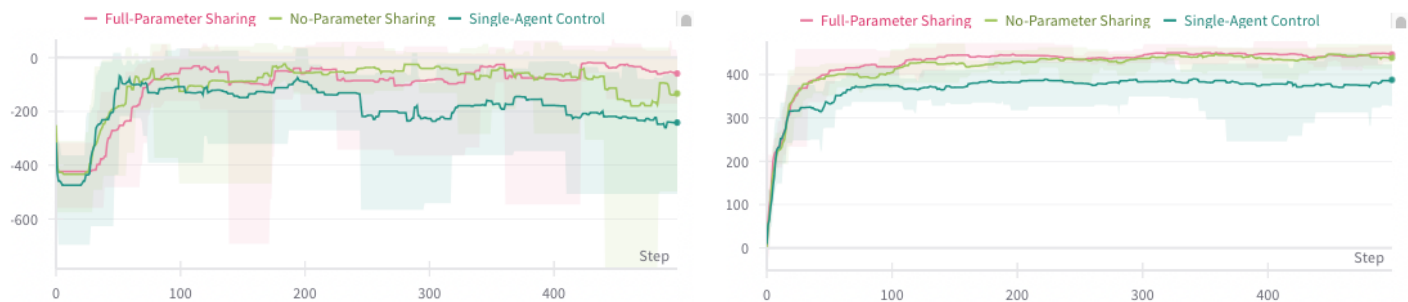
For the comparison I ran 10 instances of each algorithm using the parameters in table 2, which were values near those of the superlative performers in the previous sections. Some consideration was given to the default values of the DQN implementation. It may be noted that Mnih et al. [11] intended for 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.

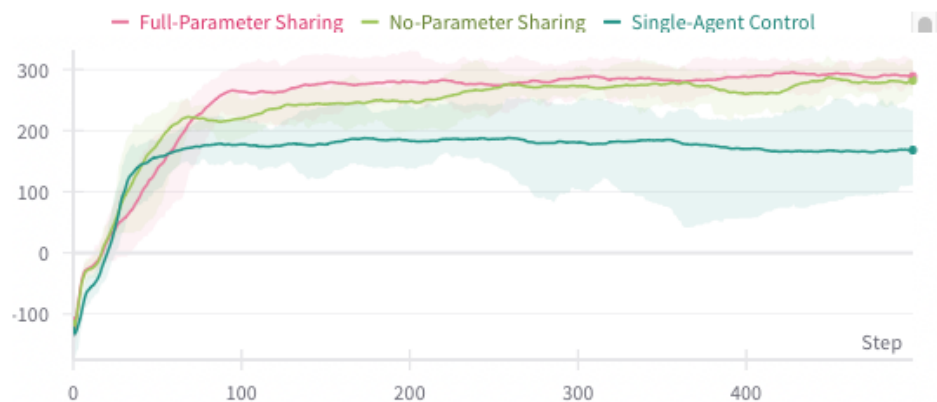
The memory utilization is about the same for each algorithm (fig. 14). Given that the algorithms use the same deep network architecture for their value functions, it reasonably follows that the memory usage would be similar, with small differences by the nature of the non-sharing networks are two separate 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

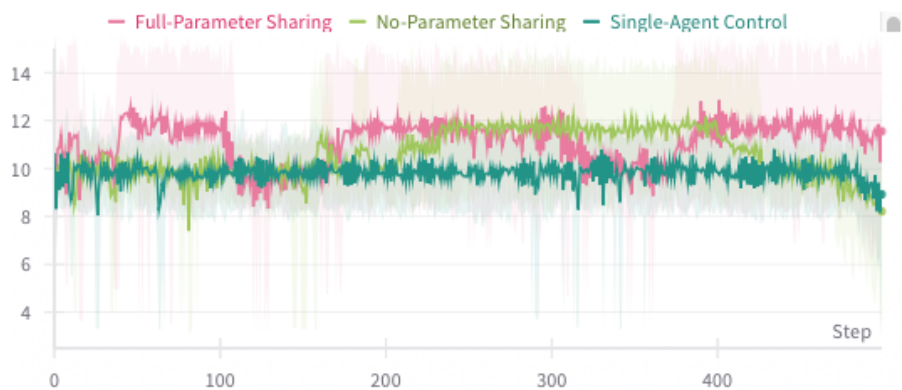


Figure 13: CPU Utilization

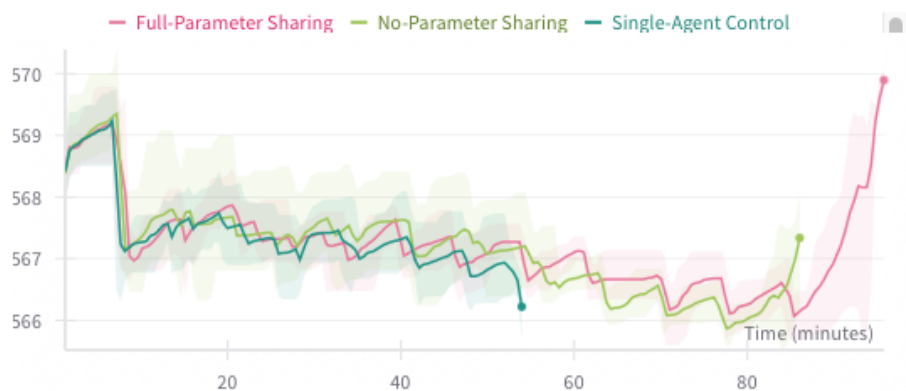


Figure 14: Memory Usage (MB)

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 or 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 how 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

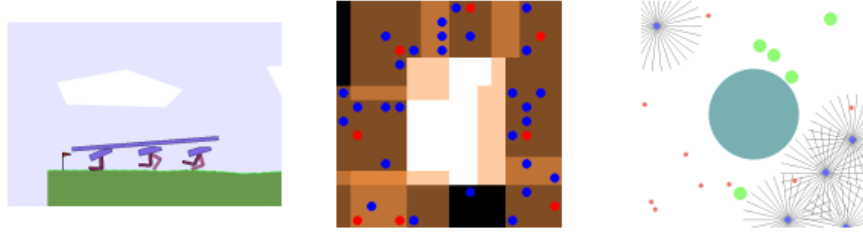


Figure 15: Multiwalker, pursuit, and waterworld environments

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

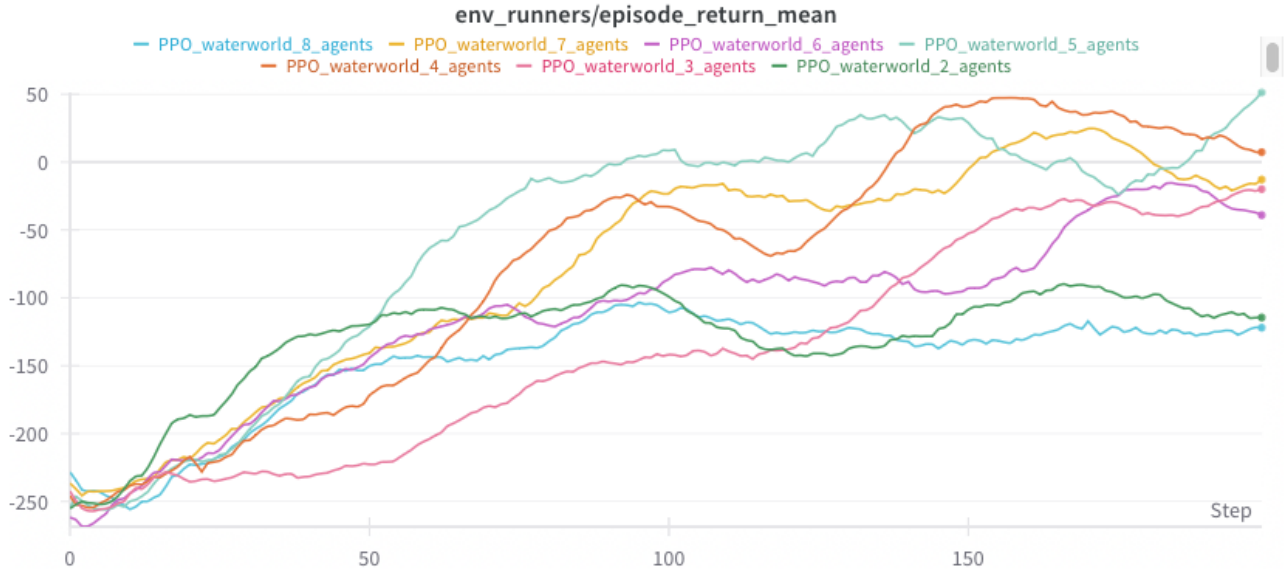


Figure 16: Mean Training Episode Return for Waterworld (Coop=2)

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.

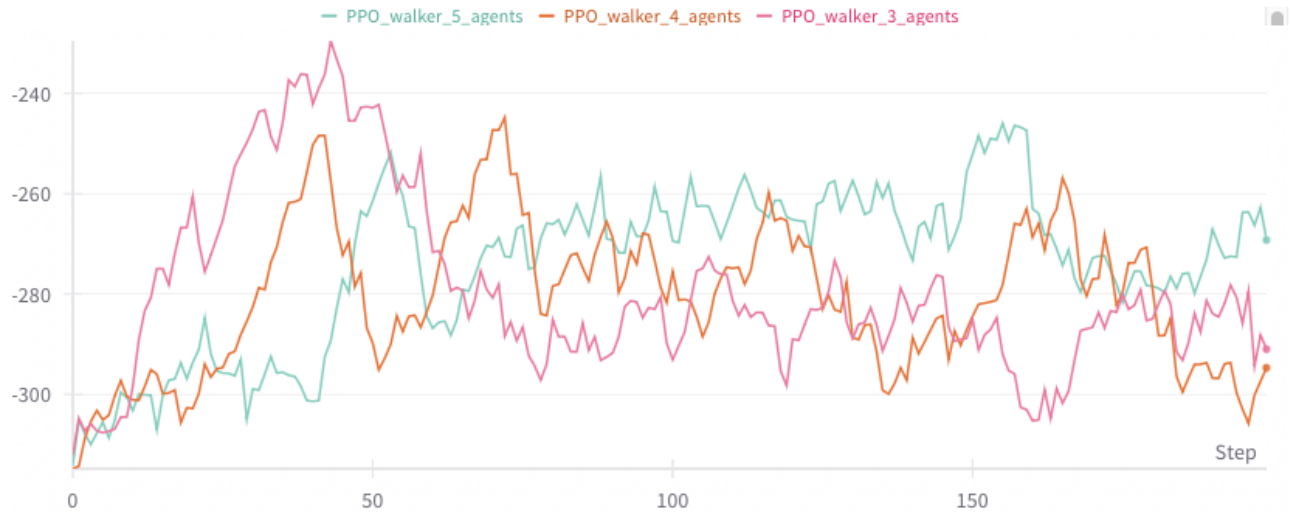


Figure 17: Mean Training Episode Return for Multiwalker

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. D biak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. J zefowicz, 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. Vinitzky, 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

```
1  """Script for Dr. Robbins Questions
2  Establishing a baseline implementation of Single Agent Lunar Lander
3  with a DQN.
4
5  See: https://gymnasium.farama.org/environments/box2d/lunar\_lander/
6  for more details on the environment.
7
8  How to run this script
9  -----
10 `python [script_name].py`
11
12 For debugging, use the following additional command line options
13 `--no-tune --num-env-runners=0`
14
15 For logging to your WandB account, use:
16 `--wandb-key=[your WandB API key] --wandb-project=[some project name]
17 --wandb-run-name=[optional: WandB run name (within the defined project)]`
18
19 """
20
21 import numpy as np
22 from gymnasium import Wrapper, spaces
23 from argparse import ArgumentParser
24
25 from ray import tune
26 from ray.rllib.core.rl_module.rl_module import SingleAgentRLModuleSpec
27 from ray.rllib.core.rl_module.marl_module import MultiAgentRLModuleSpec
28 from ray.rllib.env.wrappers.pettingzoo_env import ParallelPettingZooEnv
29 from ray.rllib.utils.test_utils import (add_rllib_example_script_args,
30                                         run_rllib_example_script_experiment)
31 from ray.tune.registry import get_trainable_cls, register_env
32
33 import multi_lander
34
35
36 parser = add_rllib_example_script_args(
37     ArgumentParser(conflict_handler='resolve'), # Resolve for num_agents
38     default_reward=600.0,
39     default_iters=500, #100, #200
40     default_timesteps=1000000, #10000, #100000
41 )
42 parser.add_argument(
43     "--control",
44     type=str,
45     choices=["Baseline", "SA", "NoPS", "FuPS"],
46     default="Baseline",
47     help="The controller method."
48     "`Baseline`: Original Lunar Lander with single agent and lander"
49     "`SA`: A single agent controls all of the landers."
50     "`NoPS`: No parameter sharing between the agents."
51     "`FuPS`: Full parameter sharing between the agents.",
52 )
53 parser.add_argument(
54     "--SA", action="store_true",
55     help="`SA`: Creates a single agent for controlling all of the agents.",
56 )
57 parser.add_argument(
58     "--NoPS", action="store_true",
```

```

59     help="``NoPS``: No parameter sharing between the agents.",
60 )
61 parser.add_argument(
62     "--FuPS", action="store_true",
63     help="``FuPS``: Full parameter sharing between the agents.",
64 )
65 parser.add_argument(
66     "--num-agents", type=int, default=2,
67     help="The number of agents",
68 )
69 parser.add_argument(
70     "--sweep", action="store_true",
71     help="Perform a parameter sweep instead of using tune",
72 )
73
74
75 class CustomWrapper(ParallelPettingZooEnv):
76     """Wraps a Parallel Petting Zoo Environment for RLlib
77
78     This wrapper is necessary to interface with rllib's workers.
79     There is a problem caused when the one agent terminates before
80     the other, which seems to pass inconsistent length episode
81     trajectories to the rollout worker(s).
82
83     It also adds the necessary ``__all__`` key to the done dictionaries.
84     """
85     def step(self, action_dict) -> tuple:
86         obs, rew, terminateds, truncateds, info = self.par_env.step(action_dict)
87
88         active = [a_id for a_id, term in terminateds.items() if term==False]
89
90         obs_s = {a_id: obs.get(a_id) for a_id in active}
91         rew_s = {a_id: rew.get(a_id) for a_id in active}
92         terminateds = {a_id: terminateds.get(a_id) for a_id in active}
93         truncateds = {a_id: truncateds.get(a_id) for a_id in active}
94
95         terminateds["__all__"] = all(terminateds.values())
96         truncateds["__all__"] = all(truncateds.values())
97         return obs_s, rew_s, terminateds, truncateds, info
98
99 # Registry is necessary for functional passing later.
100 register_env("ma-lander", lambda _: CustomWrapper(multi_lander.Parallel_Env()))
101
102
103 class SingleAgentWrapper(Wrapper):
104     """Wraps a parallel multi-agent environment for single-agent control.
105
106     *It is currently limited to agents with discrete action spaces and
107     box observation spaces with identical lengths.
108
109     This will wrap environments with an arbitrary number of agents,
110     however, the action space scales exponentially by the number of
111     agents and is likely to be the limiting factor for performance.
112     """
113     def __init__(self, *args):
114         super().__init__(*args)
115         # Concat all action spaces into a single action space
116         self.action_space = spaces.Discrete(np.prod(
117             [list(self.env.action_spaces.values())[i].n
118              for i in range(len(self.env.observation_spaces))]))

```

```

119     self._act_n = list(self.env.action_spaces.values())[0].n
120     # Concat all obs spaces into a single obs space
121     low, high = [], []
122     for i in range(len(self.env.observation_spaces)):
123         low.extend(list(self.env.observation_spaces.values())[i].low)
124         high.extend(list(self.env.observation_spaces.values())[i].high)
125     self.observation_space = spaces.Box(np.array(low), np.array(high))
126
127     def observe(self) -> tuple:
128         # Concat all obs values into a single obs tuple
129         # return sum(self.env.observe().values(), []) # By monoid
130         return np.array(list(self.env.observe().values())).flatten()
131
132     def observation(self) -> tuple:
133         return self.observe()
134
135     def reset(self, *, seed: int | None = None, options: dict | None = None) -> tuple:
136         self.env.reset(seed=seed, options=options)
137         return self.observe(), {}
138
139     def step(self, action: int) -> tuple:
140         action_list = {a: int(action / (self._act_n * i)) % self._act_n
141                        for i, a in enumerate(self.agents)}
142         obss, rews, terminations, _, _ = self.env.step(action_list)
143         obs = self.observe()
144         # Calculate reward, but drop terminated agent, else get exploding values
145         reward = sum(np.array(rews.values()) *
146                     np.invert(list(terminations.values())))
147         term = (self.env.game_over or all(terminations.values()))
148         if (term or np.any(obs < self.observation_space.low) or
149             np.any(obs > self.observation_space.high)):
150             term = True
151             obs = np.clip(obs, self.observation_space.low,
152                          self.observation_space.high)
153         return obs, reward, term, False, {}
154
155     # Registry is necessary for functional passing later.
156     register_env("sa-lander",
157                 lambda _: SingleAgentWrapper(multi_lander.Parallel_Env()))
158
159
160 if __name__ == "__main__":
161     args = parser.parse_args()
162
163     # Shared config settings for all experiments
164     base_config = (
165         get_trainable_cls("DQN")
166         .get_default_config()
167         .framework('torch')
168     )
169
170     # Set config for experiment if using Single Agent Control
171     if args.control == 'NoPS' or args.SA:
172         config = (
173             base_config
174             .environment(
175                 "sa-lander",
176                 env_config={"num_landers": args.num_agents}
177             )
178         )

```

```

179 # Set config for experiment if using No-Parameter Sharing
180 elif args.control == 'NoPS' or args.NoPS:
181     policies = {"lander_" + str(i) for i in range(args.num_agents)}
182     config = (
183         base_config
184         .multi_agent(
185             policies=policies, # 1:1 map from AgentID to ModuleID.
186             policy_mapping_fn=(lambda aid, *args, **kwargs: aid),
187         )
188         .rl_module(
189             rl_module_spec=MultiAgentRLModuleSpec(
190                 module_specs={p: SingleAgentRLModuleSpec() for p in policies},
191             ),
192         )
193         .environment(
194             "ma-lander",
195             env_config={"num_landers": args.num_agents}
196         )
197     )
198 # Set config for experiment if using Full-Parameter Sharing
199 elif args.control == 'FuPS' or args.FuPS:
200     config = (
201         base_config
202         .multi_agent(
203             policies={"p0"}, # All agents map to the same policy.
204             policy_mapping_fn=(lambda aid, *args, **kwargs: "p0"),
205         )
206         .rl_module(
207             rl_module_spec=MultiAgentRLModuleSpec(
208                 module_specs={"p0": SingleAgentRLModuleSpec()},
209             ),
210         )
211         .environment(
212             "ma-lander",
213             env_config={"num_landers": args.num_agents}
214         )
215     )
216 # Default, original lunar lander with one agent and lander
217 else:
218     config = (
219         base_config
220         .environment(
221             # env="lunar-lander"
222             "ma-lander",
223             env_config={"num_landers": 1}
224         )
225     )
226
227 # Parameter sweep settings
228 if args.sweep:
229     param_space = {
230         "adam_epsilon": tune.loguniform(1e-4, 1e-10), # 1e-8
231         "sigma0": tune.randn(0.5, 0.2), # 0.5
232         "n_step": tune.choice([2**i for i in range(5)]), # 1
233         # Update the target by \tau * policy + (1-\tau) * target_policy.
234         "tau": tune.uniform(0.0, 1.0), # 1.0,
235         # epsilon = [(0, 1.0), (10000, 0.05)] [(step, epsilon), ...]
236         # -> 1.0 at beginning, decreases to 0.05 over 10k steps
237     }
238     config = config.training(**param_space)

```

```
239         # Use Param sweep, not tune
240         #args.no_tune = True
241
242     # Call experiment runner
243     run_rllib_example_script_experiment(config, args)
```

B Multi-Agent Lunar Lander

```
1 import math
2 from typing import TYPE_CHECKING, Optional
3 import numpy as np
4
5 import gymnasium as gym
6 from gymnasium import spaces, logger
7 from gymnasium.error import DependencyNotInstalled
8 from gymnasium.utils import EzPickle
9
10 from pettingzoo import AECEnv
11 from pettingzoo.utils import agent_selector
12
13 try:
14     import Box2D
15     from Box2D.b2 import (
16         circleShape,
17         contactListener,
18         edgeShape,
19         fixtureDef,
20         polygonShape,
21         revoluteJointDef,
22     )
23 except ImportError as e:
24     raise DependencyNotInstalled(
25         'Box2D is not installed, you can install it by run `pip install swig` '
26         + 'followed by `pip install "gymnasium[box2d]"` '
27     ) from e
28
29 if TYPE_CHECKING:
30     import pygame
31
32 FPS = 50
33 SCALE = 30.0 # affects how fast the game is, forces should be adjusted as well
34 MAIN_ENGINE_POWER = 13.0
35 SIDE_ENGINE_POWER = 0.6
36
37 INITIAL_RANDOM = 1000.0 # Set 1500 to make game harder
38
39 LANDER_POLY = [(-14, +17), (-17, 0), (-17, -10), (+17, -10), (+17, 0), (+14, +17)]
40 LEG_AWAY = 20
41 LEG_DOWN = 18
42 LEG_W, LEG_H = 2, 8
43 LEG_SPRING_TORQUE = 40
44
45 SIDE_ENGINE_HEIGHT = 14
46 SIDE_ENGINE_AWAY = 12
47 MAIN_ENGINE_Y_LOCATION = 4 # The Y loc of the main engine on the lander body.
48
49 VIEWPORT_W = 600
50 VIEWPORT_H = 400
51
52
53 class ContactDetector(contactListener):
54     def __init__(self, env):
55         contactListener.__init__(self)
56         self.env = env
57
58     def BeginContact(self, contact):
```



```

59     for lander in self.env.landings:
60         if (
61             lander == contact.fixtureA.body
62             or lander == contact.fixtureB.body
63         ):
64             self.env.game_over = True
65     for leg in self.env.legs:
66         if leg in [contact.fixtureA.body, contact.fixtureB.body]:
67             leg.ground_contact = True
68
69     def EndContact(self, contact):
70         for leg in self.env.legs:
71             if leg in [contact.fixtureA.body, contact.fixtureB.body]:
72                 leg.ground_contact = False
73
74
75 class SequentialEnv(AECEnv, EzPickle):
76     r"""
77     This is a rewrite of the Farama foundation Lunar Lander environment from:
78     https://gymnasium.farama.org/environments/box2d/lunar_lander/
79     Adapted for a AEC type petting zoo environment.
80     The action space and physics remain the same.
81     """
82     metadata = {
83         "render_modes": ["human", "rgb_array"],
84         "name": "multi_lander_v0",
85         "is_parallelizable": True,
86         "render_fps": FPS,
87     }
88
89     def __init__(
90         self,
91         render_mode: Optional[str] = None,
92         continuous: bool = False,
93         gravity: float = -10.0,
94         enable_wind: bool = False,
95         wind_power: float = 15.0,
96         turbulence_power: float = 1.5,
97         num_landers: int = 2,
98         *args, **kwargs
99     ):
100         EzPickle.__init__(
101             self,
102             render_mode,
103             continuous,
104             gravity,
105             enable_wind,
106             wind_power,
107             turbulence_power,
108             *args, **kwargs
109         )
110         AECEnv.__init__(self)
111         self.render_mode = render_mode
112         self.np_random = np.random
113
114         # Value Checkers
115         assert (-12.0 < gravity and gravity < 0.0
116             ), f"gravity (current value: {gravity}) must be between -12 and 0"
117         self.gravity = gravity
118

```

```

119     if 0.0 > wind_power or wind_power > 20.0:
120         logger.warn(
121             "wind_power value is recommended to be between 0.0 and 20.0, " +
122             f"(current value: {wind_power})"
123         )
124     self.wind_power = wind_power
125
126     if 0.0 > turbulence_power or turbulence_power > 2.0:
127         logger.warn(
128             "turbulence_power value is recommended to be between 0.0 and " +
129             f"2.0, (current value: {turbulence_power})"
130         )
131     # These are bounds for position realistically the environment
132     # should have ended long before we reach more than 50% outside
133     low = np.array(
134         [
135             -2.5, -2.5,                # x,y coordinates
136             -10.0, -10.0,             # x,y velocity bounds is 5x rated speed
137             -2 * math.pi, -10.0,     # Angle, Angular Velocity
138             -0.0, -0.0,               # L,R Leg not on ground
139         ]).astype(np.float32)
140     high = np.array(
141         [
142             2.5, 5.0, # 2.5,          # x,y coordinates
143             10.0, 10.0,             # x,y velocity bounds is 5x rated speed
144             2 * math.pi, 10.0,     # Angle, Angular Velocity
145             1.0, 1.0,               # L,R Leg on ground
146         ]).astype(np.float32)
147
148     # Environmental Variables
149     self.turbulence_power = turbulence_power
150     self.enable_wind = enable_wind
151     self.screen: pygame.Surface = None
152     self.clock = None
153     self.isopen = True
154     self.world = Box2D.b2World(gravity=(0, gravity))
155     self.moon = None
156     self.particles = []
157     self.landings = []
158     self.legs = []
159     self.prev_reward = None
160     self.continuous = continuous
161
162     # Agents
163     self.possible_agents = ["lander_" + str(i) for i in range(num_landings)]
164     self.agents = self.possible_agents[:]
165     self.agent_name_mapping = dict(zip(self.agents,
166                                       list(range(self.num_agents))))
167     self._agent_selector = agent_selector(self.agents)
168
169     # Spaces
170     self.observation_spaces = dict(zip(self.agents,
171                                       [spaces.Box(low, high)] * self.num_agents))
172     if self.continuous:
173         # Action is two floats [main engine, left-right engines].
174         # Main engine: -1..0 off, 0..+1 throttle from 50% to 100% power.
175         # Engine can't work with less than 50% power.
176         # Left-right: -1.0..-0.5 fire left engine, +0.5..+1.0 fire right
177         # engine, -0.5..0.5 off
178         self._action_space = spaces.Box(-1, +1, (2,), dtype=np.float32)

```

```

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

```

```

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

```

```

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

```

```

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

```

```

194 def close(self):
195     if self.screen is not None:
196         import pygame
197         pygame.display.quit()
198         pygame.quit()
199         pygame.QUIT
200     self.isopen = False

```

```

202 def create_landers(self):
203     landers = []
204     for n in range(self.num_agents):
205         lander: Box2D.b2Body = self.world.CreateDynamicBody(
206             position=((n+1)/(self.num_agents+1) * VIEWPORT_W / SCALE,
207                       VIEWPORT_H / SCALE),
208             angle=0.0,
209             fixtures=fixtureDef(
210                 shape=polygonShape(
211                     vertices=[(x/SCALE, y/SCALE) for x, y in LANDER_POLY]),
212                 density=5.0,
213                 friction=0.1,
214                 categoryBits=0x0010,
215                 #maskBits=0x001, # uncomment for : collide only with ground
216                 restitution=0.0,
217             ), # 0.99 bouncy
218         )
219         lander.color1 = ((128), (102+75*(n))%255, (230))
220         lander.color2 = ((77), (77), (128))
221         landers.append(lander)
222     self.landerns = landers

```

```

224 legs = []
225 for n, lander in enumerate(self.landerns):
226     for i in [-1, +1]:
227         leg = self.world.CreateDynamicBody(
228             position=(lander.position[0] - i * LEG_AWAY / SCALE,
229                     lander.position[1]),
230             angle=(i * 0.05),
231             fixtures=fixtureDef(
232                 shape=polygonShape(box=(LEG_W / SCALE, LEG_H / SCALE)),
233                 density=1.0,
234                 restitution=0.0,
235                 categoryBits=0x0020,
236                 #maskBits=0x001,
237             ),
238         )

```

```

239         leg.ground_contact = False
240         leg.color1 = ((128), (102+75*(n))%255, (230))
241         leg.color2 = ((77), (77), (128))
242         rjd = revoluteJointDef(
243             bodyA=lander,
244             bodyB=leg,
245             localAnchorA=(0, 0),
246             localAnchorB=(i * LEG_AWAY / SCALE, LEG_DOWN / SCALE),
247             enableMotor=True,
248             enableLimit=True,
249             maxMotorTorque=LEG_SPRING_TORQUE,
250             motorSpeed=+0.3 * i, # low enough not to jump into the sky
251         )
252         if i == -1:
253             rjd.lowerAngle = (
254                 +0.9 - 0.5
255             ) # Valid angle of travel for the legs
256             rjd.upperAngle = +0.9
257         else:
258             rjd.lowerAngle = -0.9
259             rjd.upperAngle = -0.9 + 0.5
260         leg.joint = self.world.CreateJoint(rjd)
261         legs.append(leg)
262     self.legs = legs
263
264     def _create_particle(self, mass, x, y, ttl):
265         p = self.world.CreateDynamicBody(
266             position=(x, y),
267             angle=0.0,
268             fixtures=fixtureDef(
269                 shape=circleShape(radius=2 / SCALE, pos=(0, 0)),
270                 density=mass,
271                 friction=0.1,
272                 categoryBits=0x0100,
273                 maskBits=0x001, # collide only with ground
274                 restitution=0.3,
275             ),
276         )
277         p.ttl = ttl
278         self.particles.append(p)
279         self._clean_particles(False)
280         return p
281
282     def _clean_particles(self, all_particle):
283         while self.particles and (all_particle or self.particles[0].ttl < 0):
284             self.world.DestroyBody(self.particles.pop(0))
285
286
287     def render(self):
288         if self.render_mode is None:
289             assert self.spec is not None
290             gym.logger.warn(
291                 "You are calling render method without specifying render mode."
292                 "You can specify the render_mode at initialization, "
293                 f'e.g. gym.make("{self.spec.id}", render_mode="rgb_array")')
294             return
295
296         try:
297             import pygame
298             from pygame import gfxdraw

```

```

299 except ImportError as e:
300     raise DependencyNotInstalled(
301         'pygame is not installed, run `pip install "gymnasium[box2d]"`
302     ) from e
303
304 if self.screen is None and self.render_mode == "human":
305     pygame.init()
306     pygame.display.init()
307     self.screen = pygame.display.set_mode((VIEWPORT_W, VIEWPORT_H))
308 if self.clock is None:
309     self.clock = pygame.time.Clock()
310
311 self.surf = pygame.Surface((VIEWPORT_W, VIEWPORT_H))
312 pygame.transform.scale(self.surf, (SCALE, SCALE))
313 pygame.draw.rect(self.surf, (255, 255, 255), self.surf.get_rect())
314
315 for obj in self.particles:
316     obj.ttl -= 0.15
317     obj.color1 = (
318         int(max(0.2, 0.15 + obj.ttl) * 255),
319         int(max(0.2, 0.5 * obj.ttl) * 255),
320         int(max(0.2, 0.5 * obj.ttl) * 255),
321     )
322     obj.color2 = (
323         int(max(0.2, 0.15 + obj.ttl) * 255),
324         int(max(0.2, 0.5 * obj.ttl) * 255),
325         int(max(0.2, 0.5 * obj.ttl) * 255),
326     )
327
328 self._clean_particles(False)
329
330 for p in self.sky_polys:
331     scaled_poly = []
332     for coord in p:
333         scaled_poly.append((coord[0] * SCALE, coord[1] * SCALE))
334     pygame.draw.polygon(self.surf, (0, 0, 0), scaled_poly)
335     gfxdraw.aapolygon(self.surf, scaled_poly, (0, 0, 0))
336
337 for obj in self.particles + self.drawlist:
338     for f in obj.fixtures:
339         trans = f.body.transform
340         if type(f.shape) is circleShape:
341             pygame.draw.circle(
342                 self.surf,
343                 color=obj.color1,
344                 center=trans * f.shape.pos * SCALE,
345                 radius=f.shape.radius * SCALE,
346             )
347             pygame.draw.circle(
348                 self.surf,
349                 color=obj.color2,
350                 center=trans * f.shape.pos * SCALE,
351                 radius=f.shape.radius * SCALE,
352             )
353         else:
354             path = [trans * v * SCALE for v in f.shape.vertices]
355             pygame.draw.polygon(self.surf, color=obj.color1, points=path)
356             gfxdraw.aapolygon(self.surf, path, obj.color1)
357             pygame.draw.aalines(
358                 self.surf, color=obj.color2, points=path, closed=True)

```

359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418

```
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.landings:
        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()
```



```

419 self.world = Box2D.b2World(gravity=(0, self.gravity))
420 self.world.contactListener_keepref = ContactDetector(self)
421 self.world.contactListener = self.world.contactListener_keepref
422 self.game_over = False
423 self.prev_shaping = [None]*self.num_agents
424
425 W = VIEWPORT_W / SCALE
426 H = VIEWPORT_H / SCALE
427
428 # Create Terrain
429 CHUNKS = 11
430 height = self.np_random.uniform(0, H / 2, size=(CHUNKS + 1,))
431 chunk_x = [W / (CHUNKS - 1) * i for i in range(CHUNKS)]
432 self.helipad_x1 = chunk_x[CHUNKS // 2 - 1]
433 self.helipad_x2 = chunk_x[CHUNKS // 2 + 1]
434 self.helipad_y = H / 4
435 height[CHUNKS // 2 - 2] = self.helipad_y
436 height[CHUNKS // 2 - 1] = self.helipad_y
437 height[CHUNKS // 2 + 0] = self.helipad_y
438 height[CHUNKS // 2 + 1] = self.helipad_y
439 height[CHUNKS // 2 + 2] = self.helipad_y
440 smooth_y = [
441     0.33 * (height[i - 1] + height[i + 0] + height[i + 1])
442     for i in range(CHUNKS)
443 ]
444
445 self.moon = self.world.CreateStaticBody(
446     shapes=edgeShape(vertices=[(0, 0), (W, 0)])
447 )
448 self.sky_polys = []
449 for i in range(CHUNKS - 1):
450     p1 = (chunk_x[i], smooth_y[i])
451     p2 = (chunk_x[i + 1], smooth_y[i + 1])
452     self.moon.CreateEdgeFixture(vertices=[p1,p2],density=0,friction=0.1)
453     self.sky_polys.append([p1, p2, (p2[0], H), (p1[0], H)])
454
455 self.moon.color1 = (0.0, 0.0, 0.0)
456 self.moon.color2 = (0.0, 0.0, 0.0)
457
458 if self.enable_wind: # Initialize wind pattern based on index
459     self.wind_idx = self.np_random.integers(-9999, 9999)
460     self.torque_idx = self.np_random.integers(-9999, 9999)
461
462 self.create_landers()
463
464 for lander in self.landings:
465     lander.ApplyForceToCenter(
466         (
467             self.np_random.uniform(-INITIAL_RANDOM, INITIAL_RANDOM),
468             self.np_random.uniform(-INITIAL_RANDOM, INITIAL_RANDOM),
469         ),
470         True,
471     )
472 self.drawlist = self.legs + self.landings
473 if self.render_mode == "human": self.render()
474
475 self.agents = self.possible_agents[:]
476 self._agent_selector = agent_selector(self.agents)
477 self.agent_selection = self._agent_selector.next()
478 self.rewards = {agent: 0 for agent in self.agents}

```

```

479     self._cumulative_rewards = {agent: 0 for agent in self.agents}
480     self.terminations = {agent: False for agent in self.agents}
481     self.truncations = {agent: False for agent in self.agents}
482     self.infos = {agent: {} for agent in self.agents}
483
484     return self.observe(), None
485 # End reset()
486
487
488 def observe(self, agent=None):
489     if not agent : agent = self.agent_selection
490     id = self.agent_name_mapping[agent]
491     lander = self.landings[id]
492     pos = lander.position
493     vel = lander.linearVelocity
494
495     observation = [
496         (pos.x - VIEWPORT_W/SCALE/2) / (VIEWPORT_W/SCALE/2),
497         (pos.y - (self.helipad_y + LEG_DOWN/SCALE)) / (VIEWPORT_H/SCALE/2),
498         vel.x * (VIEWPORT_W/SCALE/2) / FPS,
499         vel.y * (VIEWPORT_H/SCALE/2) / FPS,
500         lander.angle,
501         20.0 * lander.angularVelocity / FPS,
502         1.0 if self.legs[id*2+0].ground_contact else 0.0,
503         1.0 if self.legs[id*2+1].ground_contact else 0.0,
504     ]
505     assert len(observation) == 8
506     return observation
507
508 def latest_reward_state(self, agent, state):
509     reward = 0
510     id = self.agent_name_mapping[agent]
511     shaping = (
512         -100 * np.sqrt(state[0] * state[0] + state[1] * state[1])
513         - 100 * np.sqrt(state[2] * state[2] + state[3] * state[3])
514         - 100 * abs(state[4])
515         + 10 * state[6] + 10 * state[7]
516     ) # And ten points for legs contact, the idea is if you
517        # lose contact again after landing, you get negative reward
518     if self.prev_shaping[id] is not None:
519         reward = shaping - self.prev_shaping[id]
520     self.prev_shaping[id] = shaping
521     return reward
522
523
524 def step(self, action):
525     assert self.landings is not None, "You forgot to call reset()"
526     if (
527         self.terminations[self.agent_selection]
528         or self.truncations[self.agent_selection]
529     ):
530         # If one agent has terminated this accepts a None action,
531         # which otherwise errors, handles stepping to the next agent
532         obs = self.observe(self.agent_selection)
533         self.agent_selection = self._agent_selector.next()
534         return obs, 0, True, False, {}
535
536     # Update wind and apply to the lander
537     if self.enable_wind and not any([l.ground_contact for l in self.legs]):
538         # the function used for wind is tanh(sin(2 k x) + sin(pi k x)),

```

```

539     # which is proven to never be periodic, k = 0.01
540     wind_mag = (
541         math.tanh(
542             math.sin(0.02 * self.wind_idx)
543             + (math.sin(math.pi * 0.01 * self.wind_idx))
544         ) * self.wind_power
545     )
546     self.wind_idx += 1
547
548     for lander in self.landings:
549         lander.ApplyForceToCenter(
550             (wind_mag, 0.0),
551             True,
552         )
553
554     # the function used for torque is tanh(sin(2 k x) + sin(pi k x)),
555     # which is proven to never be periodic, k = 0.01
556     torque_mag = (
557         math.tanh(
558             math.sin(0.02 * self.torque_idx)
559             + (math.sin(math.pi * 0.01 * self.torque_idx))
560         ) * self.turbulence_power
561     )
562     self.torque_idx += 1
563
564     for lander in self.landings:
565         lander.ApplyTorque(
566             torque_mag,
567             True,
568         )
569
570     # For current agent:
571     agent = self.agent_selection
572     lander = self.landings[self.agent_name_mapping[agent]]
573     # Check action validity
574     if self.continuous:
575         action = np.clip(action, -1, +1).astype(np.float64)
576     else:
577         assert self.action_spaces[agent].contains(
578             action), f"{action!r} ({type(action)}) invalid "
579
580     # Tip is the (X and Y) components of the rotation of the lander.
581     tip = (math.sin(lander.angle), math.cos(lander.angle))
582
583     # Side is the (-Y and X) components of the rotation of the lander.
584     side = (-tip[1], tip[0])
585
586     # Generate two random numbers between -1/SCALE and 1/SCALE.
587     dispersion = [self.np_random.uniform(-1.0,+1.0)/SCALE for _ in range(2)]
588
589     m_power = 0.0
590     if (self.continuous and action[0] > 0.0) or (
591         not self.continuous and action == 2
592     ):
593         # Main engine
594         if self.continuous:
595             m_power = (np.clip(action[0], 0.0, 1.0) + 1.0) * 0.5 # 0.5..1.0
596             assert m_power >= 0.5 and m_power <= 1.0
597         else:
598             m_power = 1.0

```

```

599
600 # 4 is move a bit downwards, +-2 for randomness
601 # The components of the impulse to be applied by the main engine.
602 ox = (
603     tip[0] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
604     + side[0] * dispersion[1]
605 )
606 oy = (
607     -tip[1] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
608     - side[1] * dispersion[1]
609 )
610 impulse_pos = (lander.position[0] + ox, lander.position[1] + oy)
611 if self.render_mode is not None:
612     # particles are just a decoration, with no physics impact,
613     # so don't add them when not rendering
614     p = self._create_particle(
615         3.5, # 3.5 is here to make particle speed adequate
616         impulse_pos[0],
617         impulse_pos[1],
618         m_power,
619     )
620     p.ApplyLinearImpulse(
621         (
622             ox * MAIN_ENGINE_POWER * m_power,
623             oy * MAIN_ENGINE_POWER * m_power,
624         ),
625         impulse_pos,
626         True,
627     )
628     lander.ApplyLinearImpulse(
629         (-ox * MAIN_ENGINE_POWER * m_power,
630          -oy * MAIN_ENGINE_POWER * m_power),
631         impulse_pos,
632         True,
633     )
634
635 s_power = 0.0
636 if (self.continuous and np.abs(action[1]) > 0.5) or (
637     not self.continuous and action in [1, 3]
638 ):
639     # Orientation/Side engines
640     if self.continuous:
641         direction = np.sign(action[1])
642         s_power = np.clip(np.abs(action[1]), 0.5, 1.0)
643         assert s_power >= 0.5 and s_power <= 1.0
644     else:
645         # action = 1 is left, action = 3 is right
646         direction = action - 2
647         s_power = 1.0
648
649 # The components of the impulse to be applied by the side engines.
650 ox = tip[0] * dispersion[0] + side[0] * (
651     3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
652 )
653 oy = -tip[1] * dispersion[0] - side[1] * (
654     3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
655 )
656
657 # The constant 17 is presumably meant to be SIDE_ENGINE_HEIGHT.
658 # However, SIDE_ENGINE_HEIGHT is defined as 14, causing the

```

```

659     # position of the thrust on the body of the lander to change,
660     # depending on the orientation of the lander. This results in
661     # an orientation dependent torque being applied to the lander.
662
663     impulse_pos = (
664         lander.position[0] + ox - tip[0] * 17 / SCALE,
665         lander.position[1] + oy + tip[1] * SIDE_ENGINE_HEIGHT / SCALE,
666     )
667     if self.render_mode is not None:
668         # particles are just decoration, with no impact on the physics,
669         # so don't add them when not rendering
670         p = self._create_particle(0.7, impulse_pos[0],
671                                   impulse_pos[1], s_power)
672         p.ApplyLinearImpulse(
673             (ox * SIDE_ENGINE_POWER * s_power,
674              oy * SIDE_ENGINE_POWER * s_power,),
675             impulse_pos,
676             True,
677         )
678     lander.ApplyLinearImpulse(
679         (-ox * SIDE_ENGINE_POWER * s_power,
680          -oy * SIDE_ENGINE_POWER * s_power),
681         impulse_pos,
682         True,
683     )
684
685     # Update Positions
686     self.world.Step(1.0 / FPS, 6 * 30, 2 * 30)
687
688     observation = self.observe(agent)
689     reward = self.latest_reward_state(agent, observation)
690     reward -= m_power * 0.30 # Deduct cost of fuel
691     reward -= s_power * 0.03 # Deduct cost of fuel
692
693     if self.game_over or abs(observation[0]) >= 1.0:
694         self.terminations[self.agent_selection] = True
695         reward = -100
696     if not lander.awake:
697         self.terminations[self.agent_selection] = True
698         reward = +100
699     terminated = self.terminations[self.agent_selection]
700
701     self.rewards[agent] = reward
702     self._accumulate_rewards()
703
704     self.agent_selection = self._agent_selector.next()
705
706     if self.render_mode == "human":
707         self.render()
708     # truncation=False as the time limit is handled by the `TimeLimit`
709     return observation, reward, terminated, False, {}
710
711
712 class Parallel_Env(SequentialEnv):
713     def observe(self):
714         obs_list = []
715         for i, lander in enumerate(self.landings):
716             pos = lander.position
717             vel = lander.linearVelocity
718             obs = [

```

```

719         (pos.x - VIEWPORT_W/SCALE/2) / (VIEWPORT_W/SCALE/2),
720         (pos.y - (self.helipad_y + LEG_DOWN/SCALE)) / (VIEWPORT_H/SCALE/2),
721         vel.x * (VIEWPORT_W/SCALE/2) / FPS,
722         vel.y * (VIEWPORT_H/SCALE/2) / FPS,
723         lander.angle,
724         20.0 * lander.angularVelocity / FPS,
725         1.0 if self.legs[i*2+0].ground_contact else 0.0,
726         1.0 if self.legs[i*2+1].ground_contact else 0.0,
727     ]
728     assert len(obs) == 8
729     obs_list.append(obs)
730     return dict(zip(self.possible_agents, obs_list))
731
732 def last(self):
733     return self.observe()
734
735 def step(self, action_list):
736     self._clear_rewards()
737     assert self.landings is not None, "You forgot to call reset()"
738
739     # Update wind and apply to the lander
740     if self.enable_wind and not any([l.ground_contact for l in self.legs]):
741         # the function used for wind is tanh(sin(2 k x) + sin(pi k x)),
742         # which is proven to never be periodic, k = 0.01
743         wind_mag = (
744             math.tanh(
745                 math.sin(0.02 * self.wind_idx)
746                 + (math.sin(math.pi * 0.01 * self.wind_idx))
747             ) * self.wind_power
748         )
749         self.wind_idx += 1
750
751         for lander in self.landings:
752             lander.ApplyForceToCenter(
753                 (wind_mag, 0.0),
754                 True,
755             )
756         # the function used for torque is tanh(sin(2 k x) + sin(pi k x)),
757         # which is proven to never be periodic, k = 0.01
758         torque_mag = (
759             math.tanh(
760                 math.sin(0.02 * self.torque_idx)
761                 + (math.sin(math.pi * 0.01 * self.torque_idx))
762             ) * self.turbulence_power
763         )
764         self.torque_idx += 1
765
766         for lander in self.landings:
767             lander.ApplyTorque(
768                 torque_mag,
769                 True,
770             )
771
772     #for agent in self.agents:
773     for agent, action in action_list.items():
774         # For current agent:
775         lander = self.landings[self.agent_name_mapping[agent]]
776         # Check action validity
777         if self.continuous:
778             action = np.clip(action, -1, +1).astype(np.float64)

```

```

779 else:
780     assert self.action_spaces[agent].contains(
781         action), f"{action!r} ({type(action)}) invalid "
782
783     # Tip is the (X and Y) components of the rotation of the lander.
784     tip = (math.sin(lander.angle), math.cos(lander.angle))
785
786     # Side is the (-Y and X) components of the rotation of the lander.
787     side = (-tip[1], tip[0])
788
789     # Generate two random numbers between -1/SCALE and 1/SCALE.
790     dispersion = [self.np_random.uniform(-1.0,+1.0)/SCALE for _ in range(2)]
791
792     m_power = 0.0
793     if (self.continuous and action[0] > 0.0) or (
794         not self.continuous and action == 2
795     ):
796         # Main engine
797         if self.continuous:
798             m_power = (np.clip(action[0], 0.0, 1.0) + 1.0) * 0.5 # 0.5..1.0
799             assert m_power >= 0.5 and m_power <= 1.0
800         else:
801             m_power = 1.0
802
803         # 4 is move a bit downwards, +-2 for randomness
804         # The components of the impulse to be applied by the main engine.
805         ox = (
806             tip[0] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
807             + side[0] * dispersion[1]
808         )
809         oy = (
810             -tip[1] * (MAIN_ENGINE_Y_LOCATION / SCALE + 2 * dispersion[0])
811             - side[1] * dispersion[1]
812         )
813         impulse_pos = (lander.position[0] + ox, lander.position[1] + oy)
814         if self.render_mode is not None:
815             # particles are just a decoration, with no physics impact,
816             # so don't add them when not rendering
817             p = self._create_particle(
818                 3.5, # 3.5 is here to make particle speed adequate
819                 impulse_pos[0],
820                 impulse_pos[1],
821                 m_power,
822             )
823             p.ApplyLinearImpulse(
824                 (
825                     ox * MAIN_ENGINE_POWER * m_power,
826                     oy * MAIN_ENGINE_POWER * m_power,
827                 ),
828                 impulse_pos,
829                 True,
830             )
831             lander.ApplyLinearImpulse(
832                 (-ox * MAIN_ENGINE_POWER * m_power,
833                  -oy * MAIN_ENGINE_POWER * m_power),
834                 impulse_pos,
835                 True,
836             )
837
838     s_power = 0.0

```

```

839     if (self.continuous and np.abs(action[1]) > 0.5) or (
840         not self.continuous and action in [1, 3]
841     ):
842         # Orientation/Side engines
843         if self.continuous:
844             direction = np.sign(action[1])
845             s_power = np.clip(np.abs(action[1]), 0.5, 1.0)
846             assert s_power >= 0.5 and s_power <= 1.0
847         else:
848             # action = 1 is left, action = 3 is right
849             direction = action - 2
850             s_power = 1.0
851
852         # The components of the impulse to be applied by the side engines.
853         ox = tip[0] * dispersion[0] + side[0] * (
854             3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
855         )
856         oy = -tip[1] * dispersion[0] - side[1] * (
857             3 * dispersion[1] + direction * SIDE_ENGINE_AWAY / SCALE
858         )
859
860         # The constant 17 is presumably meant to be SIDE_ENGINE_HEIGHT.
861         # However, SIDE_ENGINE_HEIGHT is defined as 14, causing the
862         # position of the thrust on the body of the lander to change,
863         # depending on the orientation of the lander. This results in
864         # an orientation dependent torque being applied to the lander.
865
866         impulse_pos = (
867             lander.position[0] + ox - tip[0] * 17 / SCALE,
868             lander.position[1] + oy + tip[1] * SIDE_ENGINE_HEIGHT / SCALE,
869         )
870         if self.render_mode is not None:
871             # particles are just decoration, with no impact on the physics,
872             # so don't add them when not rendering
873             p = self._create_particle(0.7, impulse_pos[0],
874                                     impulse_pos[1], s_power)
875             p.ApplyLinearImpulse(
876                 (ox * SIDE_ENGINE_POWER * s_power,
877                  oy * SIDE_ENGINE_POWER * s_power),
878                 impulse_pos,
879                 True,
880             )
881             lander.ApplyLinearImpulse(
882                 (-ox * SIDE_ENGINE_POWER * s_power,
883                  -oy * SIDE_ENGINE_POWER * s_power),
884                 impulse_pos,
885                 True,
886             )
887             self.rewards[agent] -= m_power * 0.30 # Deduct cost of fuel
888             self.rewards[agent] -= s_power * 0.03 # Deduct cost of fuel
889         """ End Agent Loop """
890
891         # Update Positions
892         self.world.Step(1.0 / FPS, 6 * 30, 2 * 30)
893
894         obs_list = self.observe()
895
896         for agent in self.agents:
897             obs = obs_list[agent]
898             lander = self.landings[self.agent_name_mapping[agent]]

```



```

899         self.rewards[agent] += self.latest_reward_state(agent, obs)
900         if self.game_over or abs(obs[0]) >= 1.0:
901             self.terminations[agent] = True
902             self.rewards[agent] -= 100
903         if not lander.awake:
904             self.terminations[agent] = True
905             self.rewards[agent] += 100
906
907     self._accumulate_rewards()
908
909     if self.render_mode == "human":
910         self.render()
911         # truncation=False as the time limit is handled by the `TimeLimit`
912     return obs_list, self.rewards, self.terminations, self.truncations, {}
913
914
915 def heuristic(env, s):
916     """
917     The heuristic for
918     1. Testing
919     2. Demonstration rollout.
920
921     Args:
922         env: The environment
923         s (list): The state. Attributes:
924             s[0] is the horizontal coordinate
925             s[1] is the vertical coordinate
926             s[2] is the horizontal speed
927             s[3] is the vertical speed
928             s[4] is the angle
929             s[5] is the angular speed
930             s[6] 1 if first leg has contact, else 0
931             s[7] 1 if second leg has contact, else 0
932
933     Returns:
934         a: The heuristic to be fed into the step function defined above to
935            determine the next step and reward.
936     """
937
938     angle_targ = s[0] * 0.5 + s[2] * 1.0 # angle should point towards center
939     if angle_targ > 0.4:
940         angle_targ = 0.4 # more than 0.4 radians (22 degrees) is bad
941     if angle_targ < -0.4:
942         angle_targ = -0.4
943     hover_targ = 0.55 * np.abs(
944         s[0]
945     ) # target y should be proportional to horizontal offset
946
947     angle_todo = (angle_targ - s[4]) * 0.5 - (s[5]) * 1.0
948     hover_todo = (hover_targ - s[1]) * 0.5 - (s[3]) * 0.5
949
950     if s[6] or s[7]: # legs have contact
951         angle_todo = 0
952         hover_todo = (
953             -(s[3]) * 0.5
954         ) # override to reduce fall speed, that's all we need after contact
955
956     if env.unwrapped.continuous:
957         a = np.array([hover_todo * 20 - 1, -angle_todo * 20])
958         a = np.clip(a, -1, +1)

```

```

959     else:
960         a = 0
961         if hover_todo > np.abs(angle_todo) and hover_todo > 0.05:
962             a = 2
963         elif angle_todo < -0.05:
964             a = 3
965         elif angle_todo > +0.05:
966             a = 1
967     return a
968
969 def demo_heuristic_lander(env, reps=1, seed=None, render=False):
970     total_reward = 0
971     steps = 0
972     for _ in range(reps):
973         s = env.reset(seed=seed)
974         while True:
975             s = env.last()[0]
976             a = heuristic(env, s)
977             s, r, terminated, truncated, info = env.step(a)
978             total_reward += r
979
980             if render:
981                 still_open = env.render()
982                 if still_open is False:
983                     break
984
985             if steps % 20 == 0 or terminated or truncated:
986                 print("observations:", " ".join([f"{x:+0.2f}" for x in s]))
987                 print(f"step {steps} total_reward {total_reward:+0.2f}")
988                 print(env.terminations.values())
989             steps += 1
990             # if terminated or truncated:
991             if all(env.terminations.values()):
992                 break
993     if render:
994         env.close()
995     return total_reward
996
997 def demo_parallel_heuristic(env, reps=1, seed=None, render=False):
998     total_reward = 0
999     steps = 0
1000     for _ in range(reps):
1001         _ = env.reset(seed=seed)
1002         obs = env.last()
1003         while True:
1004             actions = {agent: heuristic(env, obs[agent]) for agent in env.agents}
1005             obs, r, terminateds, truncateds, info = env.step(actions)
1006             total_reward += sum(r.values())
1007
1008             if render:
1009                 still_open = env.render()
1010                 if still_open is False:
1011                     break
1012
1013             if steps % 20 == 0 or all(terminateds) or all(truncateds):
1014                 for agent in env.agents:
1015                     print(f"{agent} observations:", " ".join([f"{x:+0.2f}" for x in obs[ag
1016                     print(f"step {steps} total_reward {total_reward:+0.2f}")
1017                     print(env.terminations.values())
1018             steps += 1

```

```

1019         # if terminated or truncated:
1020         if all(env.terminations.values()):
1021             break
1022     if render:
1023         env.close()
1024     return total_reward
1025
1026 def parallel_env(**kwargs):
1027     return Parallel_Env(**kwargs)
1028
1029 def sequential_env(**kwargs):
1030     return SequentialEnv(**kwargs)
1031
1032 def env(**kwargs):
1033     return Parallel_Env(**kwargs)
1034
1035 if __name__ == "__main__":
1036     import argparse
1037     parser = argparse.ArgumentParser(description=
1038                                     'A multi-agent version of Lunar Lander')
1039     parser.add_argument('-s', '--sequential', action="store_true",
1040                        help='Iterate as AEC environment')
1041     parser.add_argument('-n', '--num_landers', type=int, default=2,
1042                        help='number of landers')
1043     parser.add_argument('-d', '--demo_iters', type=int, default=1,
1044                        help='number of landers')
1045     args = parser.parse_args()
1046
1047     if args.sequential:
1048         _env = sequential_env(render_mode="human", num_landers=args.num_landers)
1049         demo_heuristic_lander(_env, reps=args.demo_iters, render=True)
1050     else:
1051         _env = parallel_env(render_mode="human", num_landers=args.num_landers)
1052         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.