University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

An Empirical Comparison Between Single-Agent and Multi-Agent Reinforcement Learning Algorithms in Multi-Agent Environments

ECE 499 Project Report

Ken Ming Lee
km23lee@uwaterloo.ca
20722040

Supervised by Sriram Ganapathi Subramanian and Dr. Mark Crowley

Waterloo, Ontario, Canada
August 20, 2021

Summary

Independent reinforcement learning (RL) algorithms (i.e., single-agent algorithms) are commonly used in the multi-agent setting as baselines for multi-agent algorithms to improve upon. Although independent algorithms have poor convergence properties in multi-agent settings, and have been empirically shown to perform poorly, the generalizability of neural-networks have allowed independent algorithms to overcome some of its weaknesses in certain settings. Therefore, the goal of this project is to explicitly study the empirical performances of independent algorithms in the multi-agent setting in an attempt to investigate scenarios that independent algorithms work well or fail in. We perform experiments on the three most common categories of multi-agent environments - cooperative, competitive, and mixed.

For the cooperative setting, we show that in environments where all agents have full observability and no explicit communication is required to perform well, independent algorithms can perform on par, if not better, than multi-agent algorithms. On the other hand, for cooperative environments that require communication, we show that adding recurrence to an independent algorithm's network allows it to also perform on par with other multi-agent algorithms.

For the competitive setting where both agents have full observability, we show that by performing parameter sharing and adding agent indicators to the agents' observations, independent algorithms are also able to outperform some multi-agent algorithms.

For the mixed setting, we show that due to the greedy-nature of independent algorithms, they learn to perform well independently, but fail to learn to cooperate between allies and compete with enemies.

In the appendix, we discuss some of the challenges that were faced when using popular RL libraries, and provide suggestions for future RL libraries. Lastly, we provide a list of resources for both single-agent and multi-agent RL libraries, and briefly highlight those that allow independent algorithms to work within multi-agent settings.

Contents

1 Introduction

Reinforcement learning (RL) is a general learning framework where an agent (i.e., the decision maker) learns how to act from feedback signals (i.e., rewards) obtained through interactions with the environment. For example, a robot arm can learn how to grasp objects from observations provided from a red-green-blue (RGB) camera, through repeated trial-and-error [1]. On the other hand, there are many settings in nature that require interactions between multiple entities (e.g., humans, animals and robots). For example, humans cooperate with one another in a team to bring success to an organization (cooperative), athletes represent their countries to compete in international sport events (competitive), whereas a lion pride works together to hunt a gazelle (mixed). These scenarios can be subsequently framed as RL problems through the multi-agent reinforcement learning (MARL) framework.

One of the simplest ways to extend the single-agent RL setting to the MARL setting is to assume that all agents are independent of each other. In other words, every other agent is seen as part of the environment from any agent's perspective. Unfortunately, there are well-known fundamental challenges that independent algorithms (i.e., single-agent RL algorithms) face in the multi-agent domain, such as non-stationarity, which violates the Markovian assumption of an MDP [2]. Therefore, independent algorithms have been shown to be very weak in certain scenarios [3], [4]. However, in practice, this naive framing has also shown strong empirical results in multiple environments [5]–[7]. A more recent example of this would be OpenAI Five [8]. Using independent algorithms alongside reward shaping, OpenAI Five achieved superhuman performance in Dota II, a multiplayer team-based video game that requires explicit coordination between teammates.

Despite the flaws that independent algorithms face in theory and in practice, this is still an interesting problem to study. This is because unlike most multi-agent algorithms which require observations and/or actions from other agents to learn from, independent algorithms relaxes such requirements. As a result, for specific scenarios where every agent needs to work completely independently, independent algorithms could be a viable option.

This project explicitly studies and compares empirical performance between independent and multi-agent algorithms in various multi-agent environments from PettingZoo [9], in an attempt to provide insights and potential suggestions as to whether it is possible to codify settings where independent algorithms will work, or not work well in multi-agent settings.

We show that independent algorithms can perform on par with multi-agent algorithms in the cooperative, fully observable setting; adding recurrence also allows them to perform well compared to multi-agent algorithms in environments that require communication. In the competitive setting, we show that parameter sharing alongside the addition of agent indicators allow independent algorithms to outperform some multi-agent algorithms in fully observable environments. For the mixed setting, we show that due to the greedy-nature of the independent algorithms, they learn to perform well individually, but fail to learn to cooperate between allies and compete against enemies.

2 Background Information

2.1 Reinforcement Learning

In the RL setting, an agent interacts with the environment by making sequential decisions [10]. More specifically, at every time step denoted as t, the agent observes a state s_t from the environment, and takes an action a_t. This action is executed in the environment, which returns a reward r_t and the next state s_{t+1} that are determined by the reward function $R(s_t, a_t)$ and the state transition probability $P(s_{t+1}|s_t, a_t)$ respectively. Critically, in an RL problem, $R(s_t, a_t)$ and $P(s_{t+1}|s_t, a_t)$ are part of the environment, and are usually unknown to the agent. A stochastic process satisfies the Markov property if it is memory-less, that is, the future states condition purely on the present state [11]. From the transition probability $P(s_{t+1}|s_t, a_t)$, it can be seen that the next state s_{t+1} conditions purely on the current state s_t and action a_t, therefore it satisfies the Markov property. When the interaction between the agent and the environment is defined in this way, it is called a Markov Decision Process (MDP) [12].

The goal of an RL agent is to learn a policy $\pi(a_t|s_t)$, which maps a state to an action, to maximize the expected reward it receives from the environment under the trajectory τ generated by the policy π. Mathematically, this can be encoded as the maximization of the objective function $J = \mathbb{E}_{\tau \sim P(\tau)}[G_t(\tau)]$, where:

- τ represents the trajectory taken by the agent (i.e., the sequence of states and actions), denoted by $\tau = \{s_0, a_0, s_1, a_1, \ldots, s_T, a_T\}$, where T represents the terminal state in the finite setting.
- $G_t(\tau)$ represents the sum of discounted rewards from timestep t onwards, denoted by $G_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$. A discount factor $\gamma \in (0, 1]$ is multiplied to the sum to ensure a finite reward for settings with infinite horizons.
- $P(\tau)$ represents the probability of the entire trajectory given the policy π: $P(\tau) = P(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t)$.

2.2 Multi-Agent Reinforcement Learning

The single-agent MDP framework can be extended to the MARL setting in the form of stochastic games [13]. An n-agent stochastic game can be described as follows: at every time step, each of the n agents, identified by $a \in \{1, 2, \ldots, n\}$ across all agents, takes an action u_t^a. This together forms a joint action $u_t \in U$. For notational convenience, for any agent a, we denote the joint actions of all other agents as u_t^{-a}. In this case, the reward function can either be an agent specific reward $r_t(s_t, u_t, a)$ or a team reward $r_t(s_t, u_t)$ that conditions purely on the joint action u. State transitions of the environment are determined by the state transition probability $P(s_{t+1}|s_t, u_t)$, which conditions on the state and the joint action at timestep t.

2.3 Centralized Training, Decentralized Execution

In the fully observable setting, it is technically possible to learn a centralized controller $\pi(u|s_t) : U \times S \to [0, 1]$ that maps a state to a distribution over the joint action space U. However, there are multiple fundamental challenges that arise from this. First, the joint action space U grows exponentially with the number of agents, making it intractable for environments with a large number of agents. Another challenge stems from environments where every agent only has access to its localized observations when interacting with the environment, as the lack of global states makes it difficult to perform control using a centralized controller.

Therefore, this paper is mainly focused on MARL algorithms in the Centralized Training and Decentralized Execution (CTDE) paradigm [14], [15]. In CTDE, during the control phase, rather than having a centralized controller that take actions for all agents, decentralized agents make decisions based on their individual observations. During the prediction phase, centralized training is performed, which means extra information (e.g., Markovian state) can be utilized, as long as this is not required during the control phase. This is a common setting that can be applicable to many real-world scenarios. For instance, the agents' training occurs in a simulator where agents can utilize extra state information, while the trained policies are then later deployed to environments where agents only have access to localized observations.

2.4 Cooperative, Competitive and Mixed

This paper follows the convention of classifying every MARL algorithm and environment studied into one of the three classes - cooperative, competitive, or mixed (cooperative-competitive) [16]–[19].

In the cooperative setting, agents collaborate with each other to achieve a common goal. As a result, it is very common for all agents to share the same reward function (i.e., a team goal) [20]. In such cases, the lack of individual reward signals introduces additional complexity from the multi-agent credit-assignment problem, where every agent has to deduce its own contributions from a team reward [20]. Algorithms that are studied that explicitly addresses the multi-agent credit-assignment problem includes QMIX [21] and Counterfactual Multi-Agent (COMA) Policy Gradients [3]. Additionally, the CommNet [22] extension on top of COMA is also utilized for specific cooperative environments. Other MARL algorithms that are considered for the cooperative scenario in this paper includes Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [4] and Multi-Agent Proximal Policy Optimization (MAPPO) [23], both of which learns a separate critic for every agent, allowing individual agents to have different reward functions.

In the competitive setting, agents play a zero-sum game, where an agent's gain is another agent's loss. In other words, $\sum_a r(s, u, a) = 0 \, \forall s, u$. Algorithms that are studied specifically in this paper include Deep Reinforcement Opponent Network (DRON) [24], MADDPG and MAPPO.

In the mixed or cooperative-competitive setting, environments are neither zero-sum (competitive) nor cooperative, and do not necessarily have to be general-sum either. A common setting would be environments that require every agent to cooperate with some agents, and compete with others [16]–[18]. MADDPG and MAPPO are used here, as learning a separate critic for every agent gives the algorithms flexibility to learn different behaviours for agents with different reward functions.

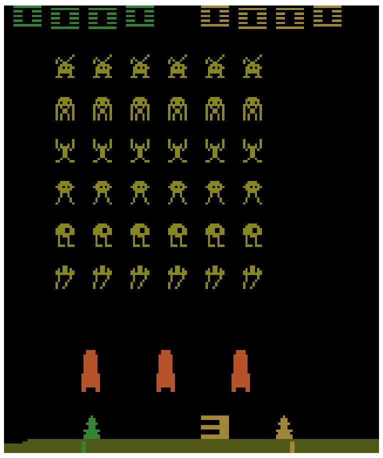
2.5 Independent Algorithms and Non-Stationarity

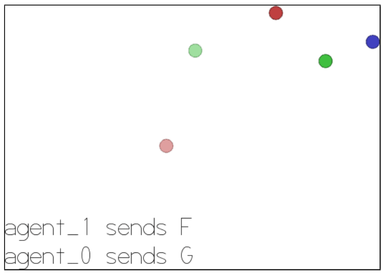
A naive application of single-agent RL to the multi-agent setting would be in the form of independent learners, where every agent treats every other agent as part of the environment and learns purely based on individual observations. While this allows independent algorithms to be extended to support multi-agent scenarios very easily, the biggest problem with this approach is that since other agents' policies are constantly changing, the environment appears *non-stationary* to any individual agent (as the transition and reward functions are conditioned on the joint action u). This violates the Markovian assumption of an MDP, making it slower to adapt to other agents' changing policies, and in certain situations, difficult to converge to a good policy [25], [26].

In this paper, independent algorithms that are studied includes Proximal Policy Optimization (PPO) [27] and Deep Q-Network (DQN) [28]. Soft Actor Critic (SAC) [29] was originally considered as well, but due to its poor performance in the discrete setting, SAC has been left out. This is described further in detail in Appendix E. Additionally, in certain cooperative environments with partial observability, we also use Deep Recurrent Q-Network (DRQN) [30], which is synonymous to DQN, with the only difference being it uses a Recurrent Neural Network (RNN) [31], [32] in addition to fully-connected ones. Since this paper studies the performance of independent algorithms in multi-agent scenarios, the aforementioned independent algorithms are executed in all 3 classes of multi-agent environments. For environments that have homogeneous state and action spaces, parameter sharing are utilized for all independent algorithms, such that only a single network is trained for all agents. Since experiences from all agents are trained simultaneously using a single network, this allows the training to be more efficient [33].

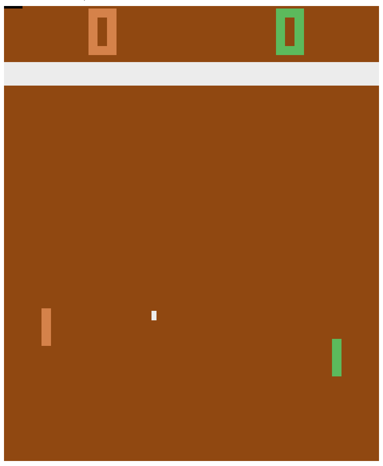
3 Experiments

3.1 Environments


(a) Space Invaders (Atari, cooperative)



(b) Simple Reference (MPE, co-operative)


(c) Pong (Atari, competitive)

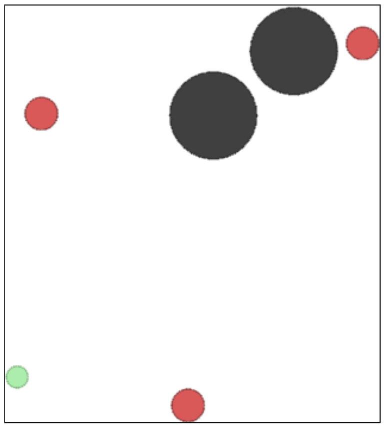

(d) Simple Tag (MPE, mixed)

Figure 1: The four environments used in the experiments. All figures were obtained from `https://pettingzoo.ml/`

The experiments were performed on multiple multi-agent environments from the PettingZoo library [9], which contains environments that include the Multi-Agent Particle Environment(MPE) [4], [34] and multi-agent versions of the Arcade Learning Environment (ALE) [35], [36], which comprise multi-agent versions of classic Atari 2600 games. These environments were chosen for their specific types (i.e., cooperative, competitive and mixed).

For the cooperative setting, experiments were performed on a modified version of the 2-player Space Invaders [35], [36], and the Simple Reference MPE environment [4], [34]. In Space Invaders, both agents share the common goal of shooting down all aliens (Figure 1a). To make Space Invaders cooperative, we removed the (positive) reward that is given to a player whenever the other player gets hit. By default, the multi-agent variant of Space Invaders rewards every agent individually. Therefore, explicit cooperation is not required to perform well, in the sense that a good combined (total) score can be achieved if both agents purely act greedily towards getting as much reward as possible for itself (i.e., shooting as many aliens as possible). We performed comparisons with another variant, where the rewards were modified such that both players always get the same (team) reward, to see how independent agents are affected by the multi-agent credit assignment problem. In contrast, in the Simple Reference environment (Figure 1b), two agents are rewarded by how close they are to their target landmark. However, the target landmark of an agent is only known by the other agent, as a result communication is required for both agents to navigate successfully to their target landmarks.

For the competitive setting, we performed experiments on a 2-player variant of the original Atari Pong environment (Figure 1c). For mixed environments, we opted for the Simple Tag MPE environment (Figure 1d), which is a predator-prey environment [34]. In this environment, the prey travels faster, and has to avoid collisions with the predators, while the 3 predators, whom travel slower, has to work together to collide with the prey. The rewards received by the prey is reciprocal to that of a predator (i.e., the prey gets a negative reward for collision, while the predators get rewarded positively), and all predators receive the same reward. Moreover, a negative reward is given to the prey to prevent it from straying too far from the predefined area. This environment is general-sum because it contains 3 predators and a single prey.

The Atari environments were originally built as video games for the Atari 2600 console, which has a memory (i.e., RAM) size of 128 bytes (1024 bits). Through the multi-agent ALE, which was built on top of Stella[1], the RAM state of the Atari machine can be extracted [35], [36]. For our experiments, we use the 128-byte Atari RAM as state inputs, rather than visual observations. This allows the algorithms to focus the learning on control rather than on both control and perception (i.e., from pixels), improving learning efficiency.

Due to time constraints, experiments performed on the Simple Reference environment ran across 5 different seeds, while for the remaining environments 3 different seeds were used. With more time, the experiments should be performed on more seeds (e.g., 10) in order to establish stronger empirical confidence bounds. Parameter sharing was also utilized for all algorithms throughout the experiments, except for in Simple Tag where the states/action spaces of the predators differ from the prey.

3.2 Preprocessing

For the MPE environments, no preprocessing was done, and default environment-parameters from PettingZoo were used for all MPE experiments.

For the Atari environments, we performed the following preprocessing - reward clipping, sticky actions, frame skipping, and no-op resets. For the Pong environment, we also concatenated a one-hot vector of the agent's index to the observations so that when using parameter sharing, the algorithm can differentiate one from the other. While preprocessing techniques are well-documented for the Atari 2600 suite to properly treat visual RGB observations [37], preprocessing techniques for RAM-state inputs are not as well-documented in the literature. Therefore, we provide appropriate explanations and justifications for the preprocessing applied in Appendix A. The number of steps per episode were also set to a limit of 200 for both Atari environments, as that yielded the best results in general (discussed further in Appendix B.2). By default, the Atari-suite has an action space size of 18. However, the effective action space for many games are smaller than that (in which case the remaining actions do nothing). For our experiments, the action spaces for both environments were shrunk to their effective action spaces in order to improve learning efficiency.

[1]`http://stella.sourceforge.net/`

3.3 Implementation

All algorithms used were based on reference implementations. Original implementations from repositories released alongside their papers were also used as much as possible.

- Implementation of independent algorithms were based on the Machin library [38]. Most independent algorithms used a neural-network with 2 hidden layers of 512×256, with hyperparameters either from the original papers or provided by default from Machin.
- Implementation of independent PPO was based on Stable Baselines3 [39]. Hyperparameters used for PPO were based on Atari hyperparameters from RL Baselines3 Zoo [40].
- Implementation of DRQN, QMIX, COMA and CommNet came from a popular public repository by the name of StarCraft [41]. Default hyperparameters were used.
- Implementation of MADDPG came from the original code implementation [4]. Default hyperparameters were used.
- Implementation of MAPPO came from the original code implementation [23]. Default MPE hyperparameters were used for all experiments.

The only notable exception was DRON; its original implementation was written in Lua [24]. Therefore, we implemented it using the Machin library [38], using the same hyperparameters as DQN. More in-depth discussions about some challenges faced during implementation and suggestions for future work can be found in Appendix B. For both DQN and DRON, the underlying DQN implementations include Double DQN [42], the dueling architecture [43] and priority experience replay buffer [44]. On the other hand, the implementation of DRQN did not use any of the aforementioned add-ons. For PPO and MAPPO, 4 parallel workers were used for all environments with homogeneous state and action spaces. Details of the hyperparameters used can be found in Appendix D.

4 Experiment Results

4.1 Cooperative

4.1.1 Simple Reference

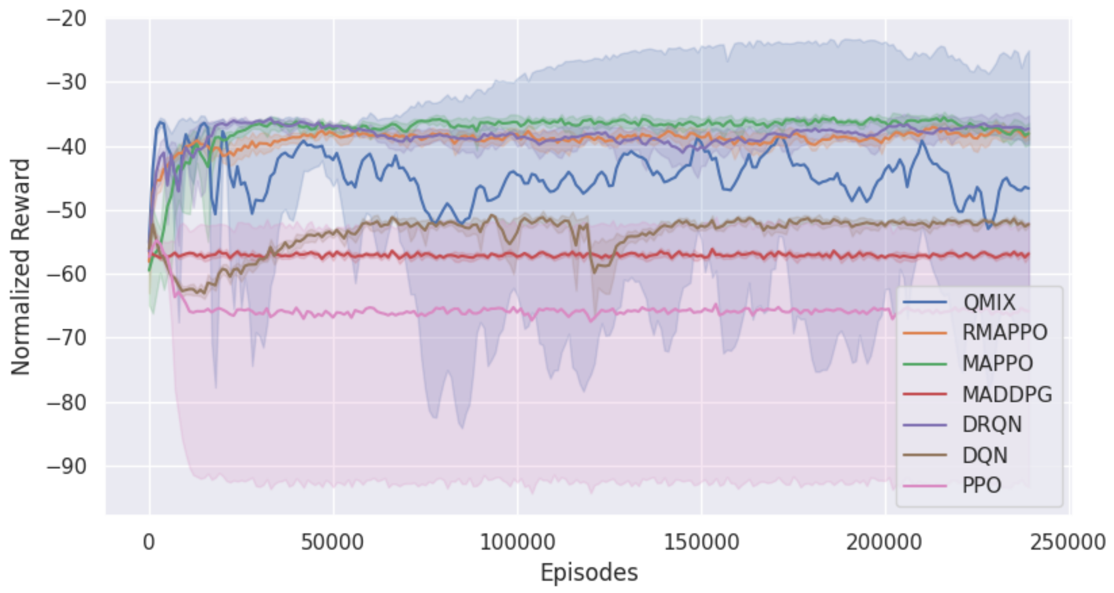


Figure 2: Comparison of various algorithms in the Simple Reference environment. For every algorithm, the solid line represents the mean reward per episode, while the shaded region represents the 95% confidence interval around the mean.

We ran the various algorithms on the Simple Reference environment for 240k episodes (6×10^6 steps). From Figure 2, it can be observed that algorithms with a temporal structure in their neural networks (e.g., RNNs), such as RMAPPO (MAPPO with a recurrent neural network), learns much more quickly compared to its purely feedforward counterpart. Similarly, the DRQN algorithm, which is identical to DQN except for the recurrent neural network that it is equipped with, outperforms DQN and is able to achieve results that are on par with other MARL algorithms, such as RMAPPO and QMIX. It can also be seen that despite having large variances from the different seeds, QMIX learns relatively quickly compared to all other algorithms.

Additionally, it can be observed that all independent algorithms converged to a suboptimal policy, except DRQN, which converged to a score that is similar to MAPPO and RMAPPO. Another interesting observation is the strong policy that MAPPO converges to, leaving the question as to whether having recurrence in the network is beneficial to this specific environment. We hypothesize that since MAPPO's centralized critic learns based on the joint observation and action of both agents, this minimizes the amount of partial observability of every agent, allowing it to learn effectively without memory. However, for independent algorithms such as DQN where the interaction between the agents cannot be explicitly learned (since all other agents are treated as part of the environment), recurrence could help agents mitigate some of the resulting partial observability. It is also noteworthy that since all algorithms utilize parameter sharing to speed up learning, further experiments could include agent indicators to the individual observations to see if that would allow independent algorithms to better learn the interactions between individual agents.

4.1.2 Space Invaders

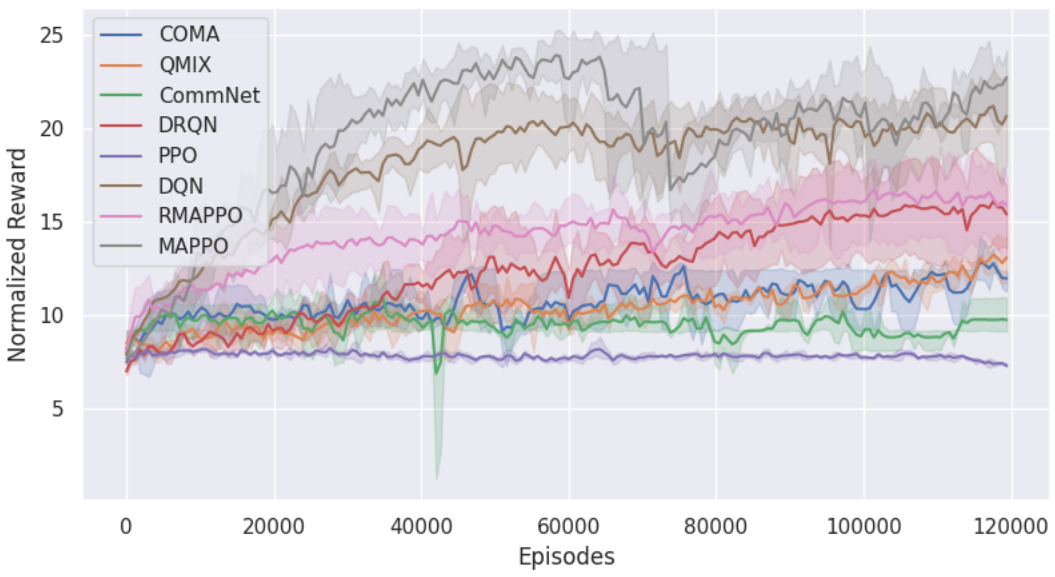


Figure 3: Comparison of various algorithms in Space Invaders

For the 2-player Space Invaders environment, MADDPG was left out due to its poor performance (Figure 4e). Since a number of cooperative MARL algorithms (e.g., COMA, QMIX) assumes that only a team reward is given, and explicitly learns the contribution of individual agents, an additional preprocessing was applied to the Space Invaders environment to make the reward equal for both agents (i.e., both agents receive the sum of their individual rewards). This setup exemplifies the multi-agent credit assignment problem, the effect of which is studied more closely in the Section 4.1.3.

From Figure 3, it can be seen that MAPPO was able to learn much more effectively compared to RMAPPO, unlike what was seen in the Simple Reference environment, where algorithms with recurrent neural networks generally performed better. This is also true for DRQN, which performed poorer than DQN. This could be the underlying reasoning behind the comparatively poorer performance of the MARL algorithms, such as QMIX, COMA and CommNet, all of which were implemented with recurrent neural networks under the CTDE scheme. Given that all agents in the Space Invaders environment have full observability over the environment, an interesting extension could be to rerun the experiments with modifications to the neural network architectures of the MARL algorithms to only use fully-connected layers without recurrence.

On the other hand, DQN's strong performance reflects the nature of the Space Invaders environment, where explicit coordination is not required to perform well, in the sense that a relatively strong joint policy can be learned by having purely greedy agents. This is especially true since there is no unit collision in the environment (i.e., agents can move past each other, and therefore an agent cannot block the other by being greedy). Additionally, since both agents have the same task, there is also less of a need for the independent algorithm to learn separate representations for individual agents from the states. This is shown in Figure 8 in Appendix A, where having an agent indicator does not yield any performance improvement for DQN on Space Invaders. PPO's weak performance, however, is a reoccurring theme from the Simple Reference environment (Figure 2). It is very likely that since DQN uses single-step bootstrapping to perform its updates, rather than over multiple steps, the extent of stochasticity of the environment faced by DQN is much lesser than PPO (the stochasticity of the environment in this case is induced by the non-stationarity). In single-agent RL, bootstrapping over a larger number of look-ahead steps reduces bias, but as a result the variance increases accordingly due to the increase in uncertainty from the predictor (e.g., critic). In the multi-agent setting, this uncertainty is compounded by the additional stochasticity of the environment that is induced by the non-stationarity. Therefore, as the number of look-ahead steps increases (i.e., as n increases in n-step TD), the harder it is to accurately estimate the value of the states, making it harder to learn. In our case, reducing the λ hyperparameter for PPO (the implementation uses Generalized Advantage Estimator (GAE)), which reduces the number of look-ahead steps, could help reduce the variance. Moreover, since only 4 parallel workers were used, it is possible that having more parallel workers alongside a larger batch size could help lower the amount of variance. Increasing the number of steps per episode of the environment could help as well, but we leave this in-depth study between on-policy and off-policy independent algorithms for MARL as a potential future work.

However, this does not explain the (stronger) performance of MAPPO compared to DQN or PPO. We hypothesize the two potential causes of this, the first is that the critic's conditioning on the joint action allowed for more advanced cooperation tactics to be learned, beyond just being greedy. The second is that by learning on the joint action, the obtained team reward is unbiased, which allows the learning process to be more efficient. This is because by conditioning on the joint action, the MAPPO's critic has full observability into the joint action that resulted in the team reward. This is in constrast to independent learners, which has to learn from a noisy team reward signal, where an agent could receive a large positive team reward even when it did nothing.

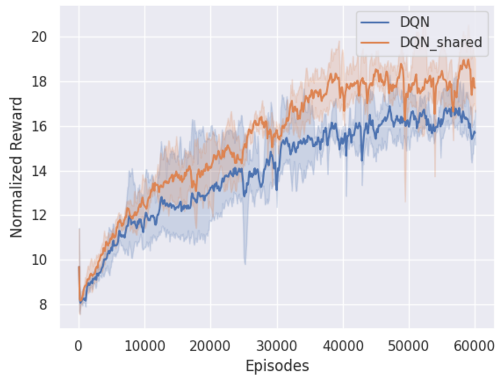
4.1.3 The Effect of Team Reward

In this section, we attempt to study the effect of how using a team reward signal, rather than individual reward signals, affect various independent and multi-agent algorithms. Also known as the multi-agent credit assignment problem, this is a core problem in the cooperative setting [19]. When only a team reward is given, this reward signal is noisy for independent algorithms, because the agent, who treats every other agents as part of the environment, does not know the actions that other agents have taken. Hence, this makes it difficult for independent algorithms to learn how the action it took contributed to the team reward signal.

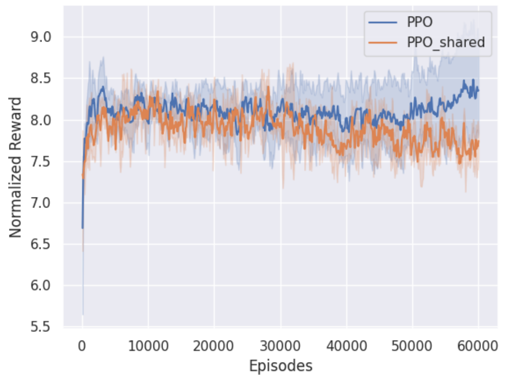
We performed the experiments on Space Invaders, in which the default agents receive individual rewards from the environment. To study the effect of the multi-agent credit assignment problem, we perform two runs per algorithm, one with only team reward, the other with individual rewards.

For multi-agent algorithms such as MAPPO (Figure 4c), RMAPPO (Figure 4d) and MADDPG (Figure 4e), having a team reward does not seem to have a large effect on the performance of the algorithms. This is as expected, since these algorithms have critics that learn from the joint action, which allows it to implicitly learn the estimated contribution of every agent.

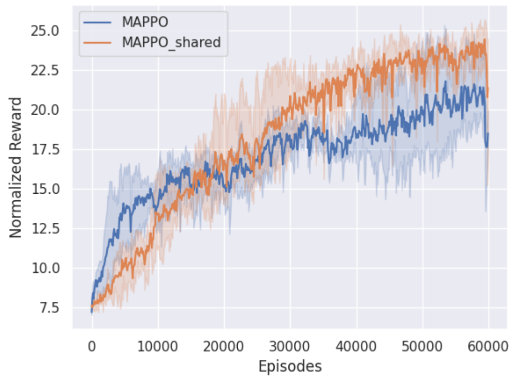
However, the same trend can be seen for the independent algorithms, where having a team reward instead of individual ones does not seem to have a very large impact on their performances. In fact, having a team reward seems to allow DQN to perform better (Figure 4a). This is likely because both agents have full observability of the state, which means that while the agent may not know what actions the other agents are taking, they could still observe the actions taken by the other agent. This is crucial for DQN, because the dueling architecture was used, which learns the value of the state $V(s_t)$ alongside the value of every action taken in the state, $Q(s_t, a_t)$. Therefore, the full observability alongside the team reward meant that a more accurate representation of $V(s)$ can be learned, which likely contributed to the higher score that is achieved while using the team reward signal. Another plausible explanation could be that since all agents receive the same reward for a given joint action, this allows the independent algorithms to relate actions from different agents that produced similar (high) rewards.



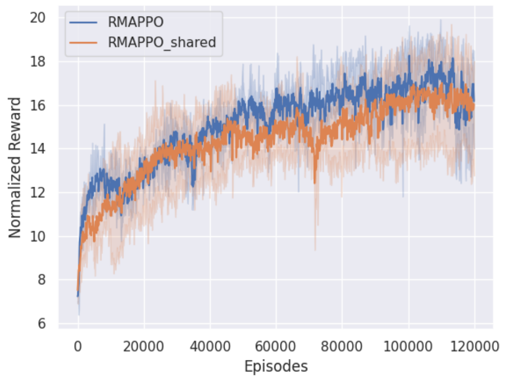
(a) DQN



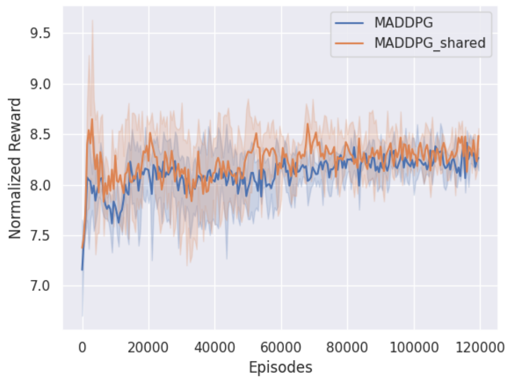
(b) PPO



(c) MAPPO



(d) RMAPPO



(e) MADDPG

Figure 4: Training curves of various algorithms in Space Invaders, comparing when individual rewards are given (blue) to when team rewards are given (denoted as `_shared` and are in orange).

4.2 Competitive

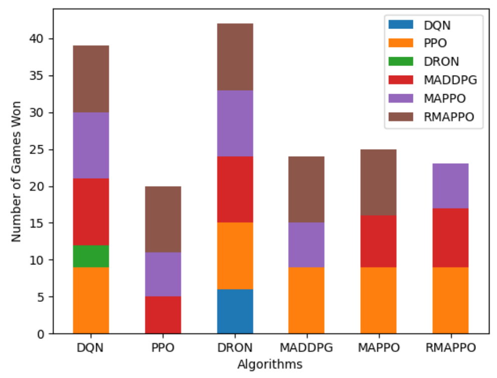
The 2-player Pong environment was used for the competitive setting. The algorithms that were evaluated includes DQN, PPO, DRON, MADDPG, MAPPO and its recurrent variant, denoted as RMAPPO. Since the observation and action spaces are homogeneous for both players, parameter sharing was utilized for all of the tested algorithms. Since Pong is a zero-sum game, we evaluated the algorithms by training all of them for 60k episodes (1.2×10^6 steps), then save their network parameters, and put them head-to-head against each other. More specifically, for each possible pair of algorithms, we put them against one another for 3 episodes. After that, their positions are swapped, and an additional 3 episodes were used to evaluate them. Swapping their positions is crucial for evaluation, because the first player (i.e., the player playing the right paddle) is always the serving player (i.e., the second (left paddle) player is always the first to hit the ball), therefore the first player always has an advantage over the second player. This advantage is further exemplified because the winning side always gets to serve subsequent openings (i.e., if the first player misses the ball, during the next round the second player will serve, so the first player would be the one to hit the ball first). This process is repeated for all seeds. An interesting property of the Pong environment is that, similar to the Space Invaders environment, the Pong environment is fully observable for both agents (i.e., both agents observe identical RAM-states). Therefore, we added an agent indicator to the RAM-state so that during training both agents can be differentiated from each other. The effect of this addition is studied more closely in Appendix A.1.

From the stacked bar charts shown in Figure 5, it can be seen that there is a similar trend across the number of games won as the first and second player (Figure 5a and 5b). DRON is consistently the best player, followed by DQN. Both of these algorithms are also the only two algorithms that have a win rate of greater than 50% for the games they have played (Figure 5d). Subsequently, observing a similar trend to the cooperative scenario, the multi-agent algorithms (MAPPO, MADDPG and RMAPPO) outperforms PPO.

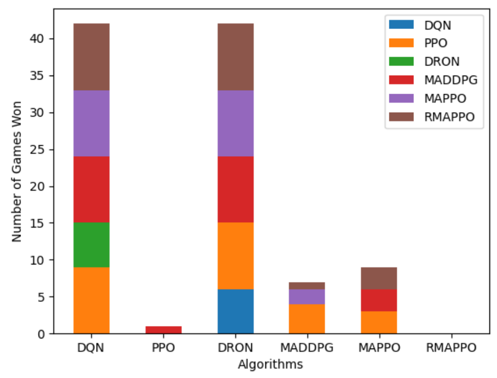
One of the interesting observations that can be made is the strong performance of independent algorithms, more specifically DQN, compared to other multi-agent algorithms. While the performance of on-policy multi-agent algorithms (MAPPO and RMAPPO) could increase with the number of parallel workers alongside better hyperparameters, we hypothesize that the strong performance of the independent algorithms in this environment could boil down to the nature of the environment. Firstly, since Pong is fully observable, critics that learn based on a joint observation of both agents do not necessarily provide any new information. Furthermore, since Pong is a highly reactive environment, an agent does not necessarily need to know the actions taken by the other agent to perform well. More specifically, since it takes several steps for the ball to propagate from a paddle to the other, an agent can learn a good policy solely by understanding how to position its paddle according to the trajectory of the ball. While learning on the joint action could allow agents to better predict the potential trajectory of the ball, this could yield diminishing returns, since there is an additional layer of complexity that will need to be learned in order for the action of the other agent to be useful (i.e., learning how to correspond a series of actions to the trajectory of the ball). These reasons could explain why the performance of the independent algorithms was on par, if not better, than some multi-agent algorithms.

On the other hand, the strong performance of DQN is indicative of the sample efficiency of the off-policy independent algorithms. Since many of the discussions regarding the potential explanation behind the better performance of off-policy independent algorithms compared to on-policy ones in the multi-agent setting that were brought up previously still holds true here, these are omitted in this section.

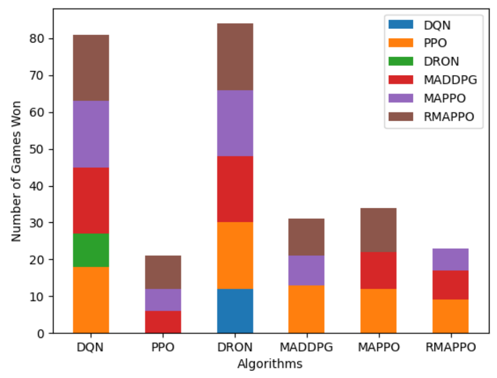
Another interesting observation that can be made is the marginal improvement that DRON has over DQN. There are two plausible reasons for this. The first is that due to the fully observable property of the Pong environment, feeding in observations from other agents do not yield any new information. The second plausible explanation is that since both algorithms utilized parameter sharing, it allows the strategies learned by both DQN agents to implicitly affect each other, leading to a more robust policy for both players. Had these algorithms train without parameter sharing, it could be likely that opponent modelling would result in a larger performance difference between both algorithms. Additionally, compared to treating other agents as part of the environment, opponent modelling allows agents to adapt more quickly to the opponent's changing strategies [24]. However, since the Pong environment is relatively simplistic compared to other more complex environments that could require more explicit strategies to be learned (e.g., deception), this could also explain the minimal improvement that DRON brings over DQN.



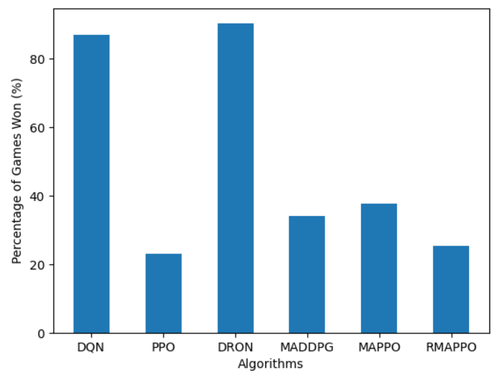
(a) Number of games won as the first player



(b) Number of games won as the second player



(c) Total number of games won



(d) Overall win rate percentage

Figure 5: Putting algorithms against each other in Pong

4.3 Mixed

For the mixed setting, the general-sum Simple Tag environment was used. Also known as the predator-prey environment, the environment consists of 3 predators and a prey. Since the predators move slower than the prey, in order for the predators to successfully hunt the prey, they have to cooperate together as a team to trap the prey. The prey is also incentivized to learn how to dodge and counter the predators' strategies, rather than constantly running further away, as the prey receives negative rewards relative to how far is it away from a predefined area. For our method of evaluation, we plot the training curves of one of the predators since all predators receive the same reward (Figure 6a), a prey (Figure 6b), and the total sum of rewards (Figure 6c), which was computed as 3×reward of a predator + reward of prey. Even though the reward function is zero-sum between a predator and a prey, since there are 3 predators in the environment, this makes the environment general-sum. Also, since the observation and action spaces differ between the predators and the prey, none of the agents have their parameters shared. Although we could have shared the parameters between the predators, we chose not to do so to ensure that bias towards the predators were not introduced (since parameter sharing between the predators would mean that they would have 3 times the amount of data to work with compared to the prey).

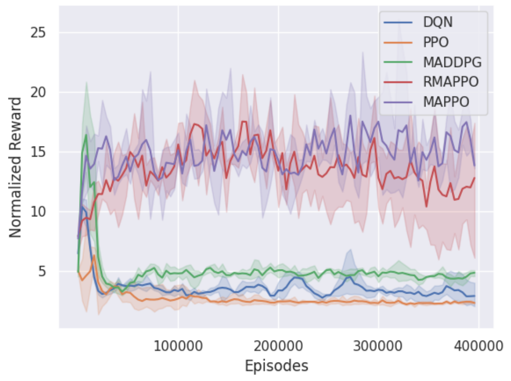
One of the first observations that stood out significantly was how different algorithms learned different final policies. For instance, based on Figure 6a and 6b, it can be seen that the policy that PPO converged onto is for the prey to run as far away from the predator as possible, and would rather incur the penalty of leaving the scene rather than getting collided with the predators. This explains the low score of the predators despite the low score of the prey.

On the other hand, since all algorithms are independent in the case of DQN, the prey has learned to be greedy, by minimizing the number of collisions with the predators. This can be seen by the strong performance achieve by the prey (Figure 6b), but weak performance by the predators (Figure 6a). Similar to PPO, since all predators are independent to each other and their parameters were not shared, they did not manage to learn how to cooperate with one another to collide with the prey.

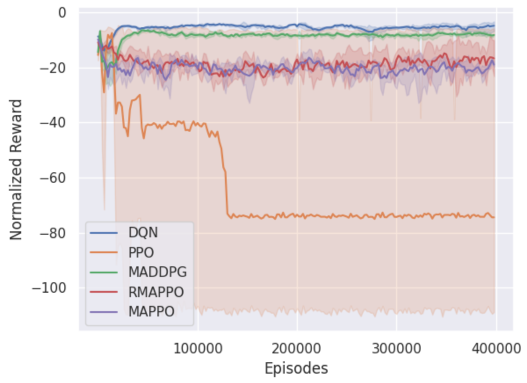
However, it is interesting to note that MADDPG seemingly converged to a similar policy to DQN, the difference being that the predators have likely learned how to better cooperate, thus getting slightly higher rewards compared to DQN's predators (Figure 6a). However, as a result, the prey's individual reward is compromised, resulting in a slightly lower score for the prey (Figure 6b).

MAPPO and RMAPPO, on the other hand, employed a different strategy. As we can observe from the comparatively noisier curves obtained from their preys and predators, it seems that there is a constant tug-of-war between the prey and the predators - as the predators learn how to cooperate better, their scores increase, which subsequently causes the prey to learn how to dodge, decreasing the predators' scores, and vice versa. Also, the policy that both MAPPO and RMAPPO converged onto is one of which the predator achieves a much higher score compared to all other algorithms, but as a result its prey has a much lower score compared to DQN and MADDPG.

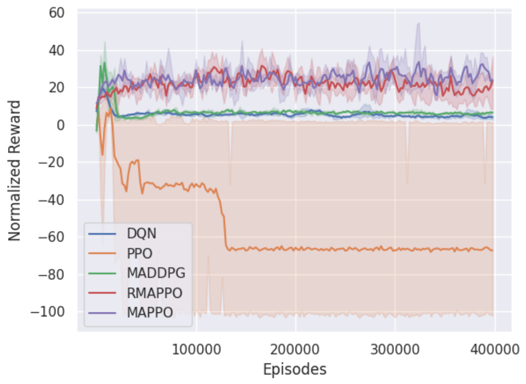
None of the agents had the objective of maximizing the total sum of rewards of all agents for two reasons: i) there are more predators than prey, and ii) the reward that a predator receives is reciprocal to that of a prey. However, summing their rewards is indicative of how well the predators have learned to cooperate relative to the prey. From Figure 6c, it can be seen that MAPPO and RMAPPO converged to a policy that achieves the highest overall total score, due to the strong performance of their predators. On the other hand, for DQN and MADDPG, the strong performance of the prey meant that there are very few collisions with the predators, and therefore their total sum of score merely hovers slightly above 0. Lastly, despite the high variance across different seeds, PPO's total sum of rewards stays negative on average, which is indicative that the predators have not learned how to properly trap the prey, neither have the prey learned to skilfully dodge the predators while staying within the play region.



(a) Reward of a predator; It should be noted that all predators obtain the same reward



(b) Reward of the prey



(c) Sum of rewards (3×predators + prey)

Figure 6: Simple Tag (predator-prey) environment

5 Conclusion

Cooperative: In the cooperative setting, for environments where individual agents have full observability such as Space Invaders, independent algorithms can perform even better than certain multi-agent algorithms. Furthermore, we showed that independent algorithms are also able to deal with the multi-agent credit assignment problem surprisingly well in environments with a relatively small number of agents, and where every agent has the same task. Similarly, in the Simple Reference environment where agents are required to communicate, which induces partial observability for independent algorithms, adding a RNN to DQN (denoted as DRQN) allowed independent algorithms to perform similarly to other multi-agent algorithms. Even though on policy algorithms like

PPO did not perform particularly well in both cooperative environments, we believe that with more careful selection of hyperparameters, it could perform better, as shown in [23]. We also discussed the significance of learning on the joint observation and action rather than individual ones, as we have shown that MAPPO performs almost equally well in the Simple Reference to DRQN, without the need for a RNN. And in the case of Space Invaders, MAPPO was able to consistently achieve the highest score amongst all other algorithms.

Competitive: In the competitive setting, we saw that DRON and DQN were able to outperform all other algorithms when playing against each other. We argued that this is very likely due to the fully observable nature of the Pong environment, in addition to the diminishing returns that learning from joint actions could yield. Furthermore, we hypothesize that through parameter sharing with agent indicators, independent algorithms are able to learn robust policies for both competing agents efficiently due to the shared representation that could be distilled into one another.

Mixed: In the predator-prey environment, we saw that since there were no parameter sharing to induce cooperation between the predators, and the predators move slower than the prey, the independent algorithms were unable to learn strong policies for their predators (i.e., the predators have not learned to cooperate effectively with each other). Conversely, in DQN we saw that its prey was able to achieve the highest score consistently, showing that the prey has learned effectively to dodge the predators while remaining within the predefined area. Interestingly, we also saw how MADDPG's training curve for its predators and prey shares resemblance to that of DQN, suggesting that it also has difficulty learning strategies for the predators to collaborate to trap the prey. MAPPO and RMAPPO, on the other hand, were the only algorithms that managed to achieve high scores for their predators, suggesting that their predators have learned how to collaborate with each other to hunt the prey. The noisiness of their graphs are also indicative of the constant tug-of-war between the prey and the predators.

6 Estimated and Actual Costs

All external libraries and code used for this project are open-source and therefore available to the public on the internet without charge. All experiments were run on the Compute Canada servers[2] using an account that was generously provided by Professor Mark Crowley. As a result, no monetary cost was incurred to the faculty for the entirety of this project.

7 Future Work

Due to time constraints of the project, there are certainly many rooms for improvement. In this section, we highlight some of these, and also provide other potential future work that could potentially bring more insights into having a broader understanding of dealing with non-stationarity and partial observability that very commonly arise in the multi-agent setting.

Since only 5 seeds were used for the Simple Reference experiments, and 3 seeds for the rest, more seeds should be run in order to give better empirical confidence bounds on the algorithms. Furthermore, since no hyperparameter tuning was used to choose the best set of hyperparameters, the performance of the algorithms could improve drastically with careful selection of hyperparameters. Both PPO and SAC performance was also quite poor on the tested environments, but it is very likely that with more tweaking and hyperparameter tuning, much better performance could be obtained. Also, more experiments on different environments, especially for the competitive and mixed setting, could bring more surprising findings. For instance, since Pong and Simple Tag are fully observable, it would be interesting to test the various algorithms on other environments with partial observability.

For cooperative environments such as Simple Reference, where goals of agents have to be communicated, we saw that DRQN was able to perform equally well as the other multi-agent algorithms. On the other hand, some multi-agent algorithms (especially those that use RNNs) performed quite poorly for Space Invaders, which is a fully observable environment. As such, future work can study

[2]`https://www.computecanada.ca/`

the extent of which recurrence can compensate for the non-stationarity that independent algorithms face in fully/partially observable environments. Similarly, further comparisons can be made against multi-agent algorithms without recurrence in fully observable environments. Also, we saw that independent algorithms were able to learn surprisingly well with just a team reward in Space Invaders. Future work could also address this to see if this was only the case for fully observable environments, or under what conditions would independent algorithms be able to cope with the multi-agent credit assignment problem in general.

For the mixed settings, with more time we could change the method of evaluating the predators and the prey for the Simple Tag environment, such that we could put different algorithms against each other to study the performance of the prey and the predators independently. An interesting investigation could be to study if policies learned from independent algorithms would allow for stronger individual performance (i.e., prey), but weaker team performance (i.e., predators).

References

[1] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection*, 2016. arXiv: `1603.02199 [cs.LG]`.

[2] S. Choi, D.-Y. Yeung, and N. Zhang, “An environment model for nonstationary reinforcement learning,” in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 2000. [Online]. Available: `https://proceedings.neurips.cc/paper/1999/file/e8d92f99edd25e2cef48eca48320a1a5-Paper.pdf`.

[3] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/11794`.

[4] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, *Multi-agent actor-critic for mixed cooperative-competitive environments*, 2020. arXiv: `1706.02275 [cs.LG]`.

[5] E. Zawadzki, A. Lipson, and K. Leyton-Brown, *Empirically evaluating multiagent learning algorithms*, 2014. arXiv: `1401.8074 [cs.GT]`.

[6] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, “Multiagent cooperation and competition with deep reinforcement learning,” *PLOS ONE*, vol. 12, no. 4, pp. 1–15, Apr. 2017. DOI: `10.1371/journal.pone.0172395`. [Online]. Available: `https://doi.org/10.1371/journal.pone.0172395`.

[7] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

[8] OpenAI, : C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, *Dota 2 with large scale deep reinforcement learning*, 2019. arXiv: `1912.06680 [cs.LG]`.

[9] J. K. Terry, B. Black, M. Jayakumar, A. Hari, R. Sullivan, L. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, C. Horsch, *et al.*, “Pettingzoo: Gym for multi-agent reinforcement learning,” *arXiv preprint arXiv:2009.14471*, 2020.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[11] A. A. Markov, “The theory of algorithms,” *Trudy Matematicheskogo Instituta Imeni VA Steklova*, vol. 42, pp. 3–375, 1954.

[12] R. BELLMAN, “A markovian decision process,” *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957, ISSN: 00959057, 19435274. [Online]. Available: `http://www.jstor.org/stable/24900506`.

[13] L. S. Shapley, “Stochastic games,” *Proceedings of the National Academy of Sciences*, vol. 39, no. 10, pp. 1095–1100, 1953, ISSN: 0027-8424. DOI: `10.1073/pnas.39.10.1095`. eprint: `https://www.pnas.org/content/39/10/1095.full.pdf`. [Online]. Available: `https://www.pnas.org/content/39/10/1095`.

[14] L. Kraemer and B. Banerjee, “Multi-agent reinforcement learning as a rehearsal for decentralized planning,” *Neurocomputing*, vol. 190, pp. 82–94, 2016, ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2016.01.031`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0925231216000783`.

[15] F. A. Oliehoek, M. T. J. Spaan, and N. A. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *CoRR*, vol. abs/1111.0062, 2011. arXiv: `1111.0062`. [Online]. Available: `http://arxiv.org/abs/1111.0062`.

[16] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008. DOI: `10.1109/TSMCC.2007.913919`.

[17] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, 2021, ISSN: 2076-3417. DOI: `10.3390/app11114948`. [Online]. Available: `https://www.mdpi.com/2076-3417/11/11/4948`.

[18] K. Zhang, Z. Yang, and T. Başar, *Multi-agent reinforcement learning: A selective overview of theories and algorithms*, 2021. arXiv: `1911.10635 [cs.LG]`.

[19] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artificial Intelligence Review*, pp. 1–49, 2021.

[20] Y.-h. Chang, T. Ho, and L. Kaelbling, "All learning is local: Multi-agent learning in global reward games," in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16, MIT Press, 2004. [Online]. Available: `https://proceedings.neurips.cc/paper/2003/file/c8067ad1937f728f51288b3eb986afaa-Paper.pdf`.

[21] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 4295–4304. [Online]. Available: `http://proceedings.mlr.press/v80/rashid18a.html`.

[22] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 2252–2260, ISBN: 9781510838819.

[23] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, *The surprising effectiveness of mappo in cooperative, multi-agent games*, 2021. arXiv: `2103.01955 [cs.LG]`.

[24] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé, "Opponent modeling in deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 1804–1813.

[25] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, "Dealing with non-stationarity in multi-agent deep reinforcement learning," *CoRR*, vol. abs/1906.04737, 2019. arXiv: `1906.04737`. [Online]. Available: `http://arxiv.org/abs/1906.04737`.

[26] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, *A survey of learning in multiagent environments: Dealing with non-stationarity*, 2019. arXiv: `1707.09183 [cs.MA]`.

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

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[29] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[30] M. Hausknecht and P. Stone, *Deep recurrent q-learning for partially observable mdps*, 2017. arXiv: `1507.06527 [cs.LG]`.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362, ISBN: 026268053X.

[32] M. I. Jordan, "Serial order: A parallel distributed processing approach. technical report, june 1985-march 1986," May 1986. [Online]. Available: `https://www.osti.gov/biblio/6910294`.

[33] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, G. Sukthankar and J. A. Rodriguez-Aguilar, Eds., Cham: Springer International Publishing, 2017, pp. 66–83, ISBN: 978-3-319-71682-4.

[34] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *arXiv preprint arXiv:1703.04908*, 2017.

[35] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013.

[36] J. K. Terry and B. Black, "Multiplayer support for the arcade learning environment," *arXiv preprint arXiv:2009.09341*, 2020.

[37] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *CoRR*, vol. abs/1709.06009, 2017. arXiv: `1709.06009`. [Online]. Available: `http://arxiv.org/abs/1709.06009`.

[38] M. Li, *Machin*, `https://github.com/iffiX/machin`, 2020.

[39] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, *Stable baselines3*, `https://github.com/DLR-RM/stable-baselines3`, 2019.

[40] A. Raffin, *Rl baselines3 zoo*, `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

[41] starry-sky6688, *Starcraft*, `https://github.com/starry-sky6688/StarCraft`, 2019.

[42] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. arXiv: `1509.06461 [cs.LG]`.

[43] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, *Dueling network architectures for deep reinforcement learning*, 2016. arXiv: `1511.06581 [cs.LG]`.

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay*, 2016. arXiv: `1511.05952 [cs.LG]`.

[45] J. Sygnowski and H. Michalewski, "Learning from the memory of atari 2600," *CoRR*, vol. abs/1605.01335, 2016. arXiv: `1605.01335`. [Online]. Available: `http://arxiv.org/abs/1605.01335`.

[46] J. K. Terry, B. Black, and A. Hari, "Supersuit: Simple microwrappers for reinforcement learning environments," *arXiv preprint arXiv:2008.08932*, 2020.

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

[48] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. arXiv: `1606.01540 [cs.LG]`.

[49] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, Jun. 2016, pp. 1928–1937. [Online]. Available: `http://proceedings.mlr.press/v48/mniha16.html`.

[50] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. G. Bellemare, and M. Bowling, “The hanabi challenge: A new frontier for ai research,” *Artificial Intelligence*, vol. 280, p. 103 216, 2020, ISSN: 0004-3702. DOI: `https://doi.org/10.1016/j.artint.2019.103216`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0004370219300116`.

[51] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. D. Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis, “OpenSpiel: A framework for reinforcement learning in games,” *CoRR*, vol. abs/1908.09453, 2019. arXiv: `1908.09453 [cs.LG]`. [Online]. Available: `http://arxiv.org/abs/1908.09453`.

[52] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, *Distributed prioritized experience replay*, 2018. arXiv: `1803.00933 [cs.LG]`.

[53] G. M. Kurtzer, V. Sochat, and M. W. Bauer, “Singularity: Scientific containers for mobility of compute,” *PLOS ONE*, vol. 12, no. 5, pp. 1–20, May 2017. DOI: `10.1371/journal.pone.0177459`. [Online]. Available: `https://doi.org/10.1371/journal.pone.0177459`.

[54] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, *Openai baselines*, `https://github.com/openai/baselines`, 2017.

[55] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable baselines*, `https://github.com/hill-a/stable-baselines`, 2018.

[56] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, *Rllib: Abstractions for distributed reinforcement learning*, 2018. arXiv: `1712.09381 [cs.AI]`.

[57] S. Huang, R. Dossa, and C. Ye, *Cleanrl: High-quality single-file implementation of deep reinforcement learning algorithms*, `https://github.com/vwxyzjn/cleanrl/`, 2020.

[58] A. Kuhnle, M. Schaarschmidt, and K. Fricke, *Tensorforce: A tensorflow library for applied reinforcement learning*, Web page, 2017. [Online]. Available: `https://github.com/tensorforce/tensorforce`.

[59] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020. [Online]. Available: `https://arxiv.org/abs/2006.00979`.

[60] J. Hu, S. Jiang, S. A. Harding, H. Wu, and S.-w. Liao, "Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning," 2021. arXiv: `2102.03479 [cs.LG]`.

[61] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, *Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks*, 2021. arXiv: `2006.07869 [cs.LG]`.

[62] A. Pretorius, K.-a. Tessera, A. P. Smit, K. Eloff, C. Formanek, S. J. Grimbly, S. Danisa, L. Francis, J. Shock, H. Kamper, W. Brink, H. Engelbrecht, A. Laterre, and K. Beguir, "Mava: A research framework for distributed multi-agent reinforcement learning," *arXiv preprint arXiv:2107.01460*, 2021. [Online]. Available: `https://arxiv.org/pdf/2107.01460.pdf`.

[63] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, and Y. Gao, *Multi-agent game abstraction via graph attention neural network*, 2019. arXiv: `1911.10715 [cs.AI]`.

[64] P. Christodoulou, *Soft actor-critic for discrete action settings*, 2019. arXiv: `1910.07207 [cs.LG]`.

Appendices

Appendix A Preprocessing on RAM-state Atari

In this section, we discuss the preprocessing techniques used and highlight their effectiveness in improving the overall performance of the algorithms.

In our experiments, the four main preprocessing techniques performed are reward clipping, sticky actions, frame skipping and no-op resets. Reward clipping ensures that the rewards at every timestep are clipped between the range of [-1, 1]. Sticky actions with a probability of 0.25 are used to inject stochasticity into the environment, and to enhance the robustness of the learned agents. We observed empirically that the addition of sticky actions and reward clipping did not have a significant effect on the learning process.

In our initial experiments, frame-skipping was left out, as we assumed that it would only be beneficial for pixel-based learning. However, this resulted in difficulty in learning for many algorithms. Clear learning was only observed from the algorithms after the addition of frame-skipping of 4 frames at every time step. This effect is consistent with the results obtained from [45] for the single-agent ALE variant. A plausible explanation of this would be that firstly, performing frame skipping speeds up learning (e.g., the five steps that the agent takes is equivalent to experiencing 20 actual frames). Secondly, unlike in environments such as MPE where feedback (rewards) are given immediately, reward signals are delayed in games with projectiles, such as Space Invaders –where the laser beam shot by an agent might only hit an enemy after a number of steps. By performing frame-skipping, the effect of delayed rewards is greatly reduced (rewards received by the agent during the x skipped frames are summed up, so they are not lost), simplifying the temporal credit-assignment problem.

Since our implementation truncates every episode to be 200-steps long, we perform a series of no-op actions at the start of every episode (these actions do not count into the step-limit per episode). Unlike in the DQN paper where initial no-ops were used to introduce randomness into the environment [28], the purpose of using no-ops in this case was to allow us to skip ahead a set number of frames at the start of every game. Space Invaders, for example, has a stall state for the first ~ 130 frames of every game, likely meant for human players to prepare for the start of the game. Removing these frames helped increase the learning efficiency for our agents. For Space Invaders and Pong, we perform no-op for the first 130 and 60 frames, respectively.

All preprocessing were performed with the help of the SuperSuit library [46].

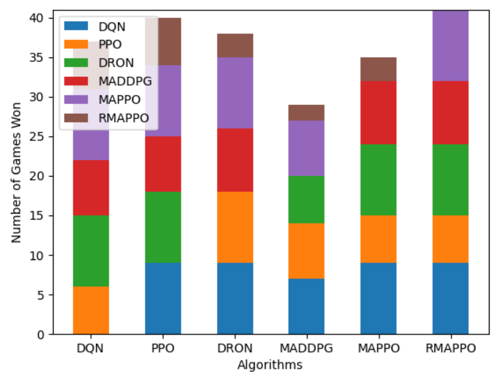
A.1 Importance of Agent Indicator

For the Pong environment, for our initial experiments, agent indicators were not concatenated to the observations of individual agents. Since Pong is fully observable, and parameter sharing was used, independent algorithms struggled to learn due to the inability to tell the difference between which paddle was the network controlling at every timestep. Given such circumstances, it was interesting to see that RMAPPO was still able to learn a reasonable policy and beat all the other algorithms (Figure 7). This is likely because RMAPPO was able to condition based on the previous observations to figure out which paddle was it controlling.

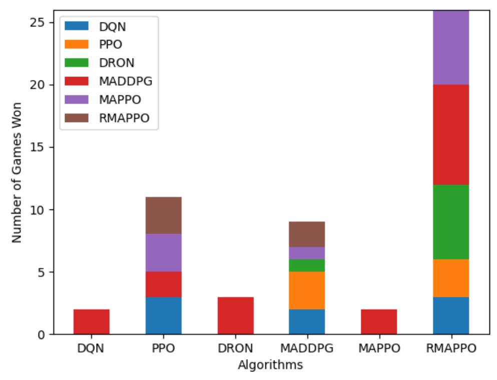
On the other hand, for the cooperative Space Invaders environment, it is interesting to observe that there is no noticeable improvement in the performance of independent algorithms when an agent indicator was added. As was previously discussed in the experimental results section, this is likely due to the similarities of both agents (i.e., both agents have the same tasks and maximize the same objectives), therefore there is less of a need to distinguish between either agent (Figure 8).

Appendix B Future Directions

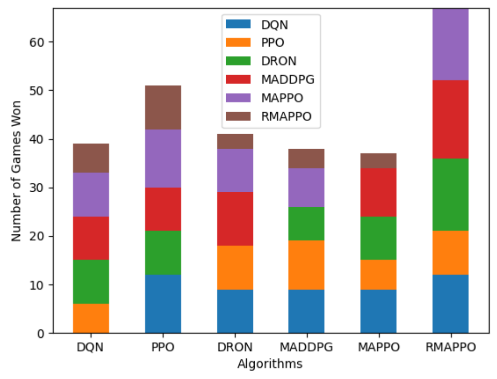
In this section, we highlight some possible future directions for making MARL algorithms more accessible, following the footsteps of the vast variety of single-agent specific RL libraries. These were distilled from some of the practical challenges that were encountered across the duration of the project.



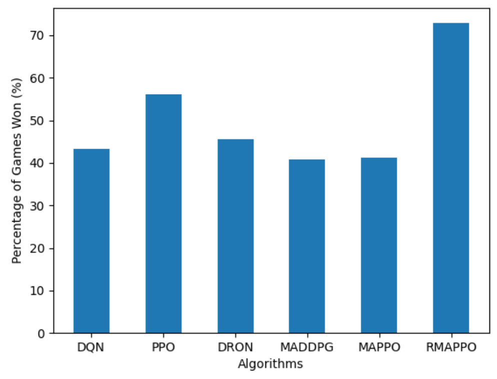
(a) Number of games won as the first player



(b) Number of games won as the second player



(c) Total number of games won



(d) Overall win rate percentage

Figure 7: Putting algorithms against each other in Pong without agent indicators

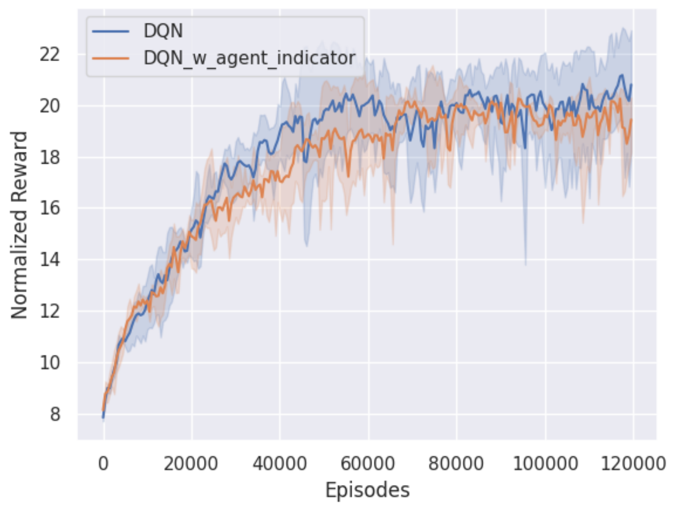


Figure 8: Comparing DQN with and without agent indicators

B.1 On Creating a MARL Library

Libraries such as Stable Baselines3 [39], that were made to make RL more accessible to researchers and the public through open, benchmarked implementations of common RL algorithms, have made it easier for the research community to replicate, identify new ideas and to build on top of existing algorithms. However, such common libraries do not yet exist for MARL algorithms (the closest we have found so far include PyMARL [47] and another library [41], but these libraries are either not actively maintained, or are not built for general-purpose use). At the time of writing, a few new upcoming libraries specifically for MARL have been released - these are listed in Appendix C.

It should be noted that the sections below are far from comprehensive on the set of challenges for implementing such a library, since unlike the single-agent case, multi-agent environments have much more variability. For instance, in the multi-agent setting, it is common for the number of agents to change (i.e., increase or decrease) across the course of an episode.

B.1.1 Implementing Independent Algorithms for Multi-Agent Environments

From the strong results that independent algorithms have shown from the various experiments in the previous sections, MARL-specific implementation libraries should also include implementations of independent algorithms in the multi-agent setting as baseline algorithms to improve upon.

The need for this is a result from the surprising incompatibility of applying commonly used baseline implementations of independent algorithms to the multi-agent setting. Even though this may seem very trivial - requiring very basic modifications to the code that handles environmental calls (i.e., `env.step()` [48]) - the nature of the modifications required makes the port nearly impossible in some cases.

For instance, a well-known single-agent RL implementation library, Stable Baselines3, recently added support for MARL through the SuperSuit library at the time of writing. This enables parameter-sharing for its parallel RL algorithms. As one of the libraries that embeds the environment code as part of the agent training code, SuperSuit enabled multi-agent support for Stable

Baselines3 by cleverly wrapping the multi-agent environment as a vectorized single-agent environment (i.e., every agent is treated as a separate worker) [46], which allows on-policy algorithms (which are commonly parallelized) like PPO and Advantage Actor Critic (A2C) [49] to indirectly support MARL environments. While this has yet to be implemented for off-policy algorithms[3], a similar approach should work as well, with the major difference being that the experience replay buffer would be shared in addition to sharing the network parameters between workers. However, this approach imposes the restriction that all agents must share their network parameters with one another. The nature of embedding the environment code within the training code also meant that workarounds like these will always be required to wrap multi-agent environments into single-agent ones.

Conversely, libraries such as Machin [38] separate the algorithms' training code from the user-customizable environmental code. Not only does this allow users to perform training on unconventional single-agent RL environments (that cannot be naturally wrapped by the OpenAI's Gym interface), but this also makes it possible to extend these algorithms to the multi-agent setting. While this approach will be more error-prone (e.g., incorrect code sequences for the environment/training), the major advantage of separating environment code from training code is the ability to customize these independent algorithms to be used in parameter-sharing, fully-independent, or in-between scenarios. Another advantage is that this approach makes the implementations environment-agnostic, especially since popular MARL environments all have their unique APIs that agents interact with (e.g., StarCraft Multi-Agent Challenge (SMAC) [47], PettingZoo/MPE [9], [34], Hanabi [50], OpenSpiel [51]).

As such, while it is definitely possible to convert existing single-agent RL libraries to support the multi-agent setting (such as Stable Baselines3), for any MARL library written from scratch, where independent algorithms are implemented as a set of baselines, *we recommend the decoupling of training code from the environment code in order for the implementations to be compatible with a wider variety of settings.*

[3]`https://github.com/DLR-RM/stable-baselines3/issues/179`

B.1.2 Assumptions About Timesteps per Episode

Another interesting and unexpected challenge that was faced during the implementation phase was the incompatibility of common MARL implementations with multi-agent ALE specifically. Interestingly, the major issue arises from the norm of having a fixed number of steps per episode for popular MARL environments, such as MPE, SMAC and Hanabi. Unlike these environments, multi-agent ALE does not have a step number limit per episode by default (similar to its single-agent counterpart), in the sense that an episode only ends when the game ends. As a result, most existing MARL implementations that expect a finite step size per episode would naturally not work well with multi-agent ALE.

Fortunately, this is a minor modification to many algorithm implementations that do not run in parallel. However, where it gets tricky is for on-policy algorithms that run in parallel, such as MAPPO. Since many of the popular MARL environments have a fixed number of step size, on-policy algorithms can run the parallel environments synchronously, such that every episode start and end together. However, if there are no step limits for the environments, parallel episodes can start and end at different times, creating the need for handling asynchronous parallel workers. While this is quite a commonality amongst many independent algorithms that run in parallel (e.g., A2C/A3C [49], PPO, Ape-X DQN [52] etc.), the issue is exemplified in MARL environments where the episode only ends when *all* agents lose the game (rather than when any one agent loses the game). Therefore, *we recommend that implementations of MARL algorithms should not make the assumption of having a set number of steps per episode in order for these implementations to be more widely applicable to other environments.*

B.1.3 Balancing Between Reproducibility, Modularity and Accessibility

To better increase the accessibility of RL research to a broader community, and to allow RL to be more applicable to other domains, there is a need for publicly released implementations to focus on the balance between reproducibility, modularity and accessibility. Reproducibility allows results from papers to be reproduced by other researchers; modularity allows the algorithms to be used in environments beyond the set of standard benchmarking environments, potentially opening the door for these algorithms to be used in other fields/domains. Accessibility does not only mean reducing the barrier of running these algorithms (e.g., on older hardware), but also tailoring the implementations to be platform-agnostic.

Since enhancing modularity commonly entails decoupling training code from environment code, which has been covered in detail previously (see Section B.1.1), in this section the focus will be on the importance of balancing between reproducibility and accessibility, since implementations that are solely focused on reproducibility may come at the cost of accessibility. For instance, it could be tempting for researchers to containerize their libraries with popular software tools such as Docker in order to allow the same results to be largely reproducible regardless of the hardware that it was run on. However, with the rise of public cloud computing systems such as Compute Canada or Google Colaboratory[4], tools/dependencies that requires administrative rights (i.e., root access) are largely inaccessible to users of such systems (mainly to ensure security of the applications). For applications that have a need for containerization, *we recommend using services like Singularity instead, which focuses specifically on maintaining security alongside reproducibility [53]. We also urge developers of future libraries to only rely on dependencies that can be installed as a non-root user (e.g., Python's Pip packages), and refrain from using packages that may not be easily accessible as much as possible.*

B.2 Robustness of Algorithms to Different Hyperparameters

One of the practical challenges that arose from the empirical nature of this study is that algorithms should be allowed to perform to the best of their abilities to have a fair comparison. Aside from the performance differences that result from the different implementations, during the process of running various experiments, we realized how much hyperparameters can affect the performance of individual algorithms.

A common practice is to perform hyperparameter search within a selected range on individual algorithms to get the best possible performance. As hyperparameter tuning was not performed for

[4]`https://colab.research.google.com/`

the experiments, default hyperparameters from the reference implementations were used as much as possible. However, the need to tweak hyperparameters was very much present for the various experiments performed for this project. For instance, aside from the usual hyperparameters related to neural networks (learning rate, batch size etc.), even something as subtle as the number of mini-batch updates for DQN after every episode can make a large difference in the performance of the algorithm (Figure 9).

Aside from the hyperparameters of the individual algorithms, environment-related hyperparameters also made a substantial difference on their performances. For instance, while performing experiments on the multi-agent variant of Space Invaders, we realized that the step limit per episode also made a large difference for multiple algorithms. In the earlier experiments, the step limits per episode were set to 800, which had an adverse impact on the performance of the algorithms, especially for multi-agent algorithms like QMIX (Figure 10b).

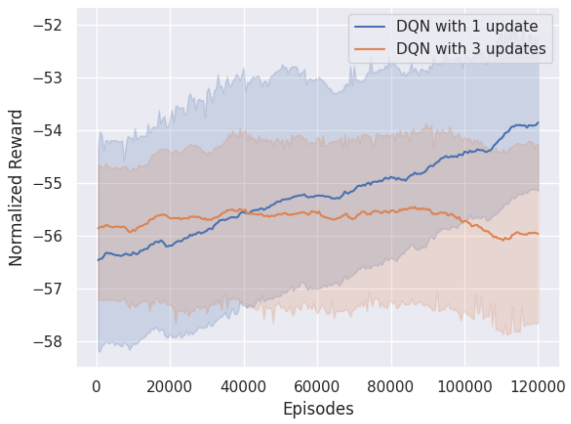
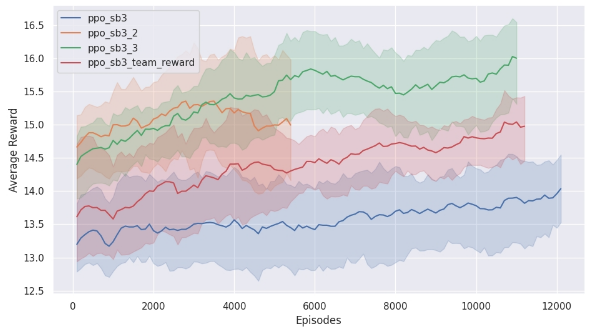
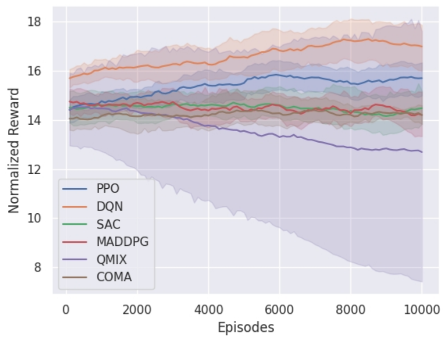


Figure 9: Learning curve of DQN with two different hyperparameters in the form of number of updates performed after every episode.



(a) Comparison of PPO on Space Invaders with different step sizes.



(b) Space Invaders with step limit of 800 per episode.

Figure 10: Effect of step-size per episode on Space Invaders

Appendix C Resources

This section aims to provide a consolidation of popular publicly available libraries of algorithm implementations, some of which were used throughout the project. While the following lists are not comprehensive, they are meant to provide more context on the categories of libraries that are available.

C.1 Single-Agent RL Libraries

The following list contains a few well-known single-agent RL libraries that provide reference implementations. Libraries in this list embeds environment code within the training code:

- OpenAI Baselines [54]
- Stable Baselines3 [39] is the next major release of stable-baselines [55], which is a fork of OpenAI's Baselines. Support for MARL environments have been recently added to a few algorithms (discussed in detail in Appendix B.1.1).
- Ray's RLlib [56] is an open source RL library that offers scalability and flexibility in training with a unified API. It should be noted that RLlib can be customized to have multi-agent

support (and do have support for several MARL algorithms), but it does have a slightly steeper learning curve compared to other libraries listed here.

- Cleanrl [57] contains single-file implementations of popular algorithms. Even though there is no explicit separation between the environment and the training code, its minimal code structure meant that only minor refactoring are required to extract and separate the environment code from the training code.

The list below are single-agent RL libraries with environment code separated from training code. This potentially allows for more flexible use cases, such as for multi-agent environments:

- Tensorforce [58]
- Deepmind's Acme [59]
- Machin [38]

C.2 Multi-Agent RL Libraries

Aside from original repositories that are released together with their paper for specific algorithms, in recent months there are new MARL-specific repositories that are released, with the same goals for single-agent RL libraries. Since some of them are relatively new and may not be as well-known as established single-agent RL libraries, we provide them in the list below, with the hope of encouraging reproducibility within the MARL community through consistent and reliable implementations.

- PyMARL [47] provides implementations for a number of cooperative MARL algorithms. Some recent extensions of it includes PyMARL2 [60] and EPyMARL [61], which adds support for additional algorithms (including a few independent algorithms).
- Mava [62] is a recently released MARL library (at time of writing) that is built on top of DeepMind's Acme.

- Although not an official library, starry-sky6688's StarCraft repository [41] provides implementations of common MARL algorithms, alongside add-on extensions such as CommNet [22] and G2ANet [63].

Appendix D Hyperparameters

Hyperparameters for DQN and DRON	
Hyperparameter	Value
fully-connected layer dimensions	512×256
optimizer	Adam
learning rate	0.001
discount factor	0.99
replay buffer size	1×10^6
batch size	256
loss function	MSE
initial epsilon	1
epsilon decay rate	0.9999
double	True
dueling	True
priority	True

Hyperparameters for PPO	
Hyperparameter	Value
fully-connected layer dimensions	64×64
number of environments	4
optimizer	Adam
number of steps	128
number of epochs	4
minibatch size	256

discount factor	0.99
GAE lambda	0.95
learning rate	linear decay from 0.00025
value function coefficient	0.05
entropy	0.01

Hyperparameters for MADDPG	
Hyperparameter	Value
fully-connected layer dimensions	64×64
optimizer	Adam
learning rate	0.01
discount factor	0.95
replay buffer size	1×10^6
batch size	1024
critic loss function	MSE
gradient clip norm	0.5

Hyperparameters for MAPPO and RMAPPO	
Hyperparameter	Value
fully-connected layer dimensions	64×64
number of environments	4
optimizer	Adam
number of epochs	10
minibatch size	1600
discount factor	0.99
GAE lambda	0.95
learning rate	0.0007
value function coefficient	0.5
gradient clip norm	0.5

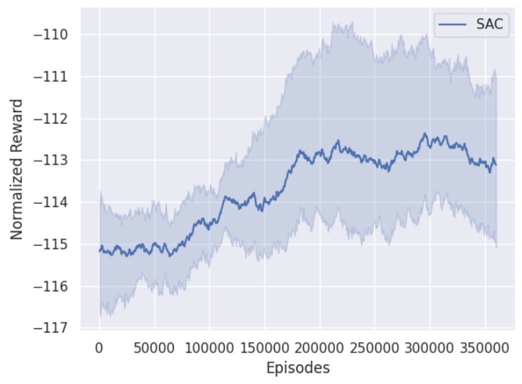
entropy	0.01
RMAPPO-specific Hyperparameters	
number of GRU layers	1
hidden state dimension	64

Hyperparameters for COMA, QMIX, DRQN and CommNet	
Hyperparameter	Value
discount factor	0.99
optimizer	RMSProp
number of GRU layers	1
hidden state dimension	64
gradient clip norm	10
batch size	256
COMA-specific Hyperparameters	
critic (fully-connected) dimension	128
actor learning rate	0.0004
critic learning rate	0.003
QMIX/DRQN-specific Hyperparameters	
hypernetwork dimension	64
learning rate	0.0005
epsilon	linear decay from 1 to 0.05
buffer size	1×10^6

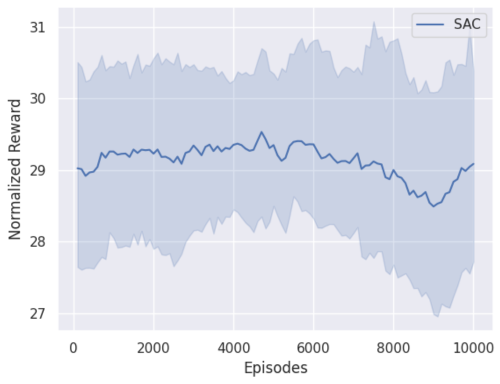
Appendix E SAC

We attempted the following modifications to discretize SAC with no avail:

- Naively squash the output with a softmax/Gumbel-softmax and sample discrete actions from it, but feed the original continuous logits into the replay buffer.



(a) MPE's Simple Reference



(b) Space Invaders

Figure 11: Performance of SAC on multiple environments across 5 different seeds

- Identical to above, but now we feed the softmax-ed probabilities into the replay buffer.
- Convert SAC's actor into a discrete actor (similar to other actor critic algorithms, such as PPO or A2C).
- Convert SAC's critic such that instead of computing $Q(s_t, a_t)$ by concatenating s_t and a_t as an input to the neural network (which is similar to other continuous-action algorithms such as DDPG and TD3), the architecture of the Q network is changed to have n outputs, where n is the number of actions (which can be typically found in Q networks of discrete-action algorithms, such as DQN).

Although discretized SAC has shown great results in certain discrete environments [64], due to time constraints of the project, SAC was left out from the comparisons due to the subpar performance obtained. With more tweaking and some hyperparameter optimization, it would be highly likely to see much better results.