

## A Preprocessing

In this section, we detail the preprocessing techniques used on the multi-agent Atari (RAM-state) environments. 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. On the other hand, adding a frame-skipping of 4 gave a significant improvement to the empirical performance of the algorithms. This is likely due to the increase in simulation-speed (e.g., the five steps that the agent takes is equivalent to experiencing 20 actual frames). Another subtle side-effect of this preprocessing is that 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 [27], 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  $\sim 130$  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 using the SuperSuit library [36].

## B Importance of Agent Indicator

In this section we list some interesting findings regarding the addition of agent indicators to independent algorithms when performing parameter sharing.

Interestingly, in both cooperative environments, there is no noticeable improvement in the performance of independent algorithms when an agent indicator was added (Fig. 6). As was previously discussed in the experimental results section, in the case of Space Invaders, this is likely due to the similarities of both agents in terms of their tasks, and their representations (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. On the other hand, for the Simple Reference environment, it is very likely that the agent indicators did not make a noticeable difference because of the partially observable nature of the environment; adding recurrence would result it a much more significant difference instead.

Conversely, for the Pong environment, even though it is also fully observable (akin to Space Invaders), the representation of both agents are not interchangeable. Utilizing parameter sharing without agent indicators, all algorithms struggled to learn due to the inability to tell which paddle was it controlling at every timestep. The only exception was RMAPPO (Fig. 7), which was able to condition on the sequence of previous observations to figure out which paddle was it controlling.

## C Implementation Details

The following list contains the sources of the reference implementations for the various algorithms:

- Implementation of DQN and DRON were based on the Machin library [37].
- Implementation of independent PPO was based on Stable Baselines3 [38].
- Implementation of DRQN, QMIX, COMA and CommNet came from a popular public repository by the name of StarCraft [39].
- Implementation of MADDPG came from the original code implementation [8].
- Implementation of MAPPO came from the original code implementation [17].

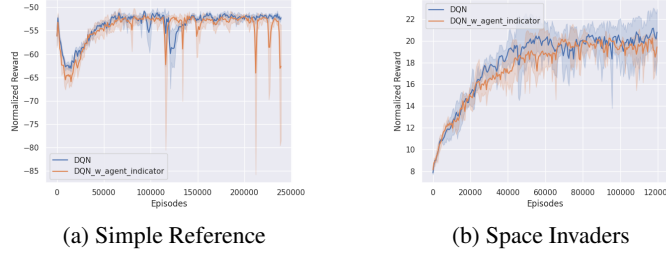


Figure 6: Comparing DQN with (blue) and without (orange) agent indicators in Simple Reference and Space Invaders environment

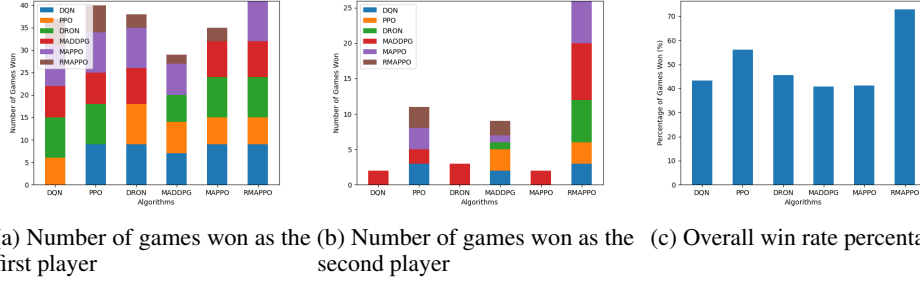


Figure 7: Putting algorithms against each other in Pong without agent indicators across 3 seeds

For both DQN and DRON, the underlying DQN implementations include Double DQN [40], the dueling architecture [41] and priority experience replay buffer [42]. 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.

### C.1 Hyperparameters

In this section, we specify the hyperparameters used for all algorithms used throughout the experiments.

Table 1: 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	$1 \times 10^6$
batch size	256
loss function	MSE
initial epsilon	1
epsilon decay rate	0.9999
double	True
dueling	True
priority	True

Table 2: Hyperparameters for PPO

Hyperparameter	Value
fully-connected layer dimensions	$64 \times 64$
number of environments	4
optimizer	Adam
number of steps	episode length
number of epochs	10
minibatch size	episode length*# of agents*4
discount factor	0.99
GAE lambda	0.95
learning rate	0.0007
value function coefficient	0.5
clip	0.2
entropy	0.01

Table 3: Hyperparameters for MADDPG

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

Table 4: Hyperparameters for MAPPO and RMAPPO

Hyperparameter	Value
fully-connected layer dimensions	$64 \times 64$
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
clip	0.2
entropy	0.01
RMAPPO-specific Hyperparameters	
number of GRU layers	1
hidden state dimension	64

Table 5: 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	$1 \times 10^6$