Responsibility Statement

The authors bear all responsibility in case of violation of rights in the proposed environments and the included benchmark. All the proposed environments use an MIT license for their source code.

A New Multi-Agent Reinforcement Learning Environments

As part of this work, we developed and open-sourced two novel MARL environments focusing on cooperation and sparse-rewards. In this supplementary section, we will provide details about both added environments including our motivation for creating them, accessibility and licensing, installation information, a description of their interface and code snippets to get started as well as further details on their observations, reward functions and dynamics. For high-level descriptions, see Sections 3.4 and 3.5 and see Appendix C for more information on the specific tasks used for the benchmark.

A.1 Motivation

Environments for MARL evaluation are scattered and few environments have been established as standard benchmarking problems. Most notably, the Starcraft Multi-Agent Challenge (SMAC) [Samvelyan et al., 2019] and Multi-Agent Particle Environment (MPE) [Mordatch and Abbeel, 2017] are prominent examples for MARL environments. While more environments exist, they often represent different types of games such as turn-based board games [Bard et al., 2020] or contain image frames as observations [Johnson et al., 2016, Beattie et al., 2016, Resnick et al., 2018]. Environments with these properties often require further solutions not specific to MARL. Additionally, we found that the core challenge of exploration, for which a multitude of environments exist for single-agent RL research (e.g. Atari games such as Montezuma's Revenge [Bellemare et al., 2013]), is underrepresented in MARL environments. The Level-Based Foraging and Multi-Robot Warehouse environments aim to represent sparse-reward hard exploration problems which require significant cooperation across agents. Both environments are flexible in their configurations to enable partial- or full-observability, fully-cooperative or mixed reward settings and allow faster training than a multitude of other environments (see Appendix A.8 for a comparison of simulation speeds across all environments used in the benchmark).

A.2 Accessibility and Licensing

Both environments are publicly available on GitHub under the following links.

Table 4: GitHub repositories for Level-Based Foraging and Multi-Robot Warehouse environments.

Level-Based Foraging	`https://github.com/uoe-agents/lb-foraging`
Multi-Robot Warehouse	`https://github.com/uoe-agents/robotic-warehouse`

The environments are licensed under the MIT license which can be found in the `LICENSE` file within respective repositories. Both environments will be supported and maintained by the authors as needed.

A.3 Installation

Our novel environments can be installed as Python packages. We recommend users to setup a virtual environment to manage packages and dependencies for individual projects, e.g. using `venv` or Anaconda. Then the code repository can be cloned using git and installed as a package using the Python package manager `pip` as follows at the example of the Multi-Robot Warehouse environment:

```
$ git clone git@github.com:uoe-agents/robotic-warehouse.git
$ cd robotic-warehouse
$ pip install -e .
```

In order to install the Level-Based Foraging environment, execute the following, similar code:

```
$git clone git@github.com:uoe-agents/lb-foraging.git
$cd lb-foraging
$pip install -e .
```

A.4 Environment Interface

Both environments follow the interface framework of OpenAI's Gym. Below, we will piece-by-piece explain the interface and commands needed to interact with the installed environment within Python.

Package import First, the installed package and `gym` (installed above as dependency) need to be imported (shown at the example of the Multi-Robot Warehouse):

```
import gym
import robotic_warehouse
```

Environment creation Following the import of the required packages, the environment can be instantiated. A large selection of configurations for both environments are already registered to `gym` upon importing the package. These can simply be created:

```
env = gym.make("rware-tiny-2ag-v1")
```

For an overview over the naming of these environment configuration names, see Appendix A.5.

Start an episode A new episode within the environment can be started with the following command:

```
obs = env.reset()
```

This function returns the initial state or observation of the environment, indicating s_0.

Environment steps In order to interact with the environment, an action for each agent should be provided. In the case of the `"rware-tiny-2ag-v1"`, there are two agents in the environment. Therefore, the environment expects to receive an action for all agents. In order to gain insight into the shape of expected actions or received observations, the respective spaces of the environment can be inspected:

```
env.action_space  # Tuple(Discrete(5), Discrete(5))
env.observation_space  # Tuple(Box(XX,), Box(XX,))
```

Using the action space of the environment, we can sample random valid actions and take a step in the environment:

```
actions = env.action_space.sample()  # the action space can be sampled
print(actions)                       # e.g. (1, 0)
next_obs, reward, done, info = env.step(actions)

print(done)    # [False, False]
print(reward)  # [0.0, 0.0]
```

Note, that for these multi-agent environments, actions are a `list` or `tuple` containing individual actions for all agents. Similarly, the received values are also lists of observations at the next timestep (`next_obs`), rewards for the completed transition (`reward`), flags indicating whether the episode has terminated (`done`) and an additional dictionary (`info`) which may contain meta-information on the transition or episode.

Rendering Both novel environments support rendering to visualise states and inspect agents' behaviour. See Figure 4 for exemplary visualisations of these environments.

```
env.render()
```

Single episode We can put all these steps together for a script which executes a single episode using random actions and rendering the environment:

```
import gym
import robotic_warehouse

env = gym.make("rware-tiny-2ag-v1")

obs = env.reset()
done = [False] * env.n_agents

while not all(done):
    actions = env.action_space.sample()
    next_obs, reward, done, info = env.step(actions)
    env.render()

env.close()
```

A.5 Environment Naming

In this section, we will briefly outline the configuration possibilities for Level-Based Foraging and Multi-Robot Warehouse environments.

Level-Based Foraging For the Level-Based Foraging environment, we can create an environment as follows

```
env = gym.make("Foraging<obs>-<x_size>x<y_size>-<n_agents>p-<food>f<force_c>-v1")
```

with the following options for each field:

- <obs>: This optional field can either be empty ("") or indicate a partially observable task with visibility radius of two fields ("-2s").
- <x_size>: This field indicates the horizontal size of the environment map and can by default take any values between 5 and 20.
- <y_size>: This field indicates the vertical size of the environment map and can by default take any values between 5 and 20. It should be noted, that upon import only environments with square dimensions (<x_size> = <y_size>) are registered and ready for creation.
- <n_agents>: This field indicates the number of agents within the environment. By default, any values between 2 and 5 are automatically registered.
- <food>: This field indicates the number of food items scattered within the environment. It can take any values between 1 and 10 by default.
- <force_c>: This optional field can either be empty ("") or indicate a task with only "cooperative food" ("-coop". In the latter case, the environment will only contain food of a level such that all agents have to cooperate in order to pick the food up. This mode should only be used with up to four agents.

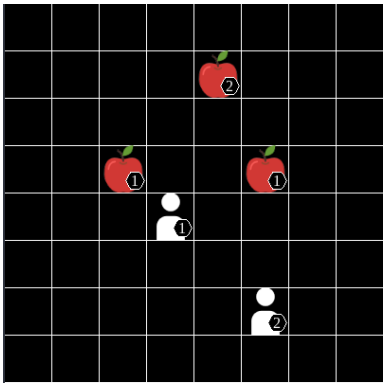
In order to register environments of more different configurations, see the current registration.

Multi-Robot Warehouse For the Multi-Robot Warehouse environment, we can create an environment as follows

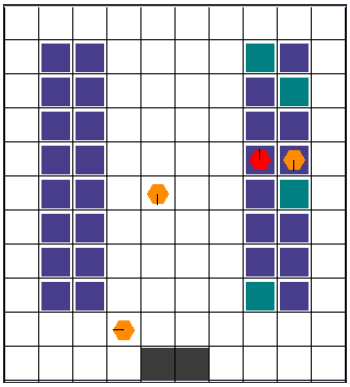
```
env = gym.make("rware-<size>-<num_agents>ag<diff>-v1")
```

with the following options for each field:

- <size>: This field represents the size of the warehouse. By default the size can take on four values: "tiny", "small", "medium" and "large". These size identifiers define the number of rows and columns of groups of shelves within the warehouse and set these to be (1, 3), (2, 3), (2, 5) and (3, 5) respectively. By default, each group of shelves consists of 16 shelves organised in a 8×2 grid.



(a) `Foraging-8x8-2p-3f-v1`



(b) `rware-tiny-4ag-v1`

Figure 4: Environment renderings matching observations for (a) Level-Based Foraging and (b) Multi-Robot Warehouse.

- <`num_agents`>: This field indicates the number of agents and can by default take any values between 1 and 20.
- <`diff`>: This optional field can indicate changes in the difficulty of the environment given by the total number of requests at a time. Agents have to collect and deliver specific requested shelves. By default, there are N requests at each point in time with N being the number of agents. With this field, the number of requests can be set to half (`"-hard"`) or double (`"-easy"`) the number of agents.

Additionally, by default agents observe only fields within immediate grids next to their location and episodes terminate after 500 steps. For a more extensive set of configurations, including variations of visibility, see the full registration.

A.6 Environment Details - Level-Based Foraging

Below, we will provide additional details to observations, actions and dynamics for the Level-Based Foraging environment.

Observations As seen above, agents receive observations at each timestep which correspond to the full state of the environment or a partial view in the case of partial observability. Below, we will outline a realistic observation at the example of the `Foraging-8x8-2p-3f-v1` task visualised in Figure 4a:

```
(
    array([1., 4., 2., 3., 2., 1., 3., 5., 1., 6., 6., 2., 4., 4., 1.],
          dtype=float32),
    array([1., 4., 2., 3., 2., 1., 3., 5., 1., 4., 4., 1., 6., 6., 2.],
          dtype=float32)
)
```

The observation consists of two arrays, each corresponding to the observation of one of the two agents within the environment. Within that vector, triplets of the form (x, y, level) are written. Specifically, the first three (number of food items in the environment) triplets for a total of 9 elements contain the x and y coordinates and level of each food item, and the following two (number of agents) triplets have the respective values for each agent. The coordinates always start from the bottom left square in the observability radius of the agent. When food items or agents are not visible, either because they are outside of the observable radius or the food has been picked up, then the respective values are replaced with (-1, -1, 0).

Actions In Level-Based Foraging environments, each agent has six possible discrete actions to choose at each timestep:

$$A^i = \{\text{Noop, Move North, Move South, Move West, Move East, Pickup}\}$$

While the first action corresponds to the action simply staying within its grid and the last action being used to pickup nearby food, the remaining four actions encode discrete 2D navigation. Upon choosing these actions, the agent moves a single cell in the chosen direction within the grid.

Rewards Agents only receive non-zero reward for picking up food within the Level-Based Foraging environment. The reward for picking up food depends on the level of the collected food as well as the level of each contributing agent. The reward of agent i for picking up food is defined as

$$r^i = \frac{FoodLevel * AgentLevel}{\sum FoodLevels \sum LoadingAgentsLevel}$$

with normalisation. Rewards are normalised to ensure that the sum of all agents' returns on a solved episode equals to one.

Dynamics The transition function of the Level-Based Foraging domain is fairly straightforward. Agents transition to cells following their movement based on chosen actions. Furthermore, agents successfully collect food as long as the sum of the levels of all loading agents is greater or equal to the level of the food.

A.7 Environment Details - Multi-Robot Warehouse

Now, we will provide additional details to observations, actions and dynamics for the Multi-Robot Warehouse environment.

Observations Agent observations for the Multi-Robot Warehouse environment are defined to be partially observable and contain all information about cells in the immediate proximity of an agent. By default, each agent observes information within a 3×3 square centred on the agent, but the visibility range can be modified as an environment parameter. Agents observe their own location, rotation and load, the location and rotation of other observed agents as well as nearby shelves with information whether those are requested or not. For the precise details, see the example below matching the state visualised in Figure 4b:

```
(
    array([8., 3., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1.,
           0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           1., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1.,
           1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 0.,
           0., 0., 0.], dtype=float32),
    array([4., 9., 0., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0.], dtype=float32),
    ...
)
```

Again, each element of the list is the observation that corresponds to each of the agents. The first three values of the vectors correspond to the agent itself with the x and y coordinates, and a value of "1" or "0" depending on whether the agent is currently carrying a shelf. The next four values are a one-hot encoding of the direction the current agent is facing (up/down/left/right respectively for each item in the one-hot encoding). Then, a single value of "1" or "0" represents whether the agent is currently in a location that acts as a path and therefore not allowed to place shelves on that location. The rest of the values can be split into 9 groups of 7 elements, each group corresponding to a square in the observation radius (3x3 centred around the agent - can be increased from the default for more visibility). In this group, the elements are only ones and zeros and correspond to: agent exists in the square, one-hot encoding (4 elements) of direction of agent (if exists), shelf exists in the square, shelf (if exists) is currently requested to be delivered to the goal location.

Actions The action space within the Multi-Robot Warehouse environment is very similar to the Level-Based Foraging domain with four discrete actions to choose from:

$$A^i = \{\text{Turn Left, Turn Right, Move Forward, Load/ Unload Shelf}\}$$

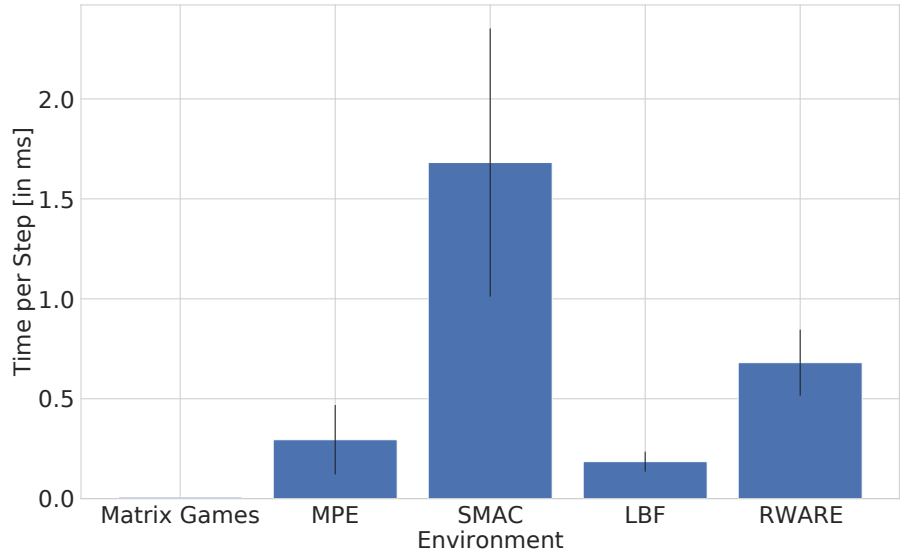


Figure 5: Mean simulation time per step for all environments. Bars indicate standard deviations of simulation speed across all tasks within the environments.

Agents also have to navigate a 2D grid-world, but they are only able to move forward or rotate to change their orientation in $90\,^{\circ}$ steps in either direction. Agents are unable to move upon cells which are already occupied. Besides movement, agents are only able to load and unload shelves. Loading shelves is only possible when the agent is located at the location of an unloaded shelf. Similarly, agents are only able to unload a currently loaded shelf on a location where no shelf is located but is within a group where initially a shelf has been stored.

Rewards At each time, a set number of shelves R is requested. Agents are rewarded with a reward of 1 for successfully delivering a requested shelf to a goal location at the bottom of the warehouse. A significant challenge in this environment is for agents to successfully deliver requested shelves but also finding an empty location to return the previously delivered shelf. Without unloading the previously delivered shelf, the agent is unable to collect new requested shelves. Having multiple steps between deliveries leads to a very sparse reward signal.

Dynamics Agents move through the grid-world as expected given their chosen actions. Whenever multiple agents collide, i.e. they attempt to move to the same location, movement is resolved in a way to maximise mobility. When two or more agents attempt to move to the same location, we prioritise agents to move which also blocks other agents. Otherwise, the selection is done arbitrarily. Additionally, it is worth noting that a shelf is uniformly sampled and added to the list of currently requested shelves whenever a previously requested shelf is delivered to a goal location. Therefore, R requested shelves are constantly ensured. Note that R directly affects the difficulty of the environment. A small R, especially on a larger grid, dramatically affects the sparsity of the reward and thus makes exploration more difficult as randomly delivering a requested shelf becomes increasingly improbable.

A.8 Environments Simulation Speed Comparison

In order to demonstrate and compare simulation speed of all considered environments within the conducted benchmark, we simulate 10,000 environments steps using random action selection. Environments were simulated without rendering to represent conditions at training or evaluation time. Total time and time per step are reported in Table 5. Additionally, mean simulation time per step with standard deviations across for all environments are reported in Figure 5.

Table 5: Simulation time for 10,000 steps and time per step in all 25 tasks.

	Tasks	Number of agents	Total time [in s]	Time per step [in ms]
Matrix Games	Climbing	2	0.079	0.008
	Penalty k=0	2	0.079	0.008
	Penalty k=-25	2	0.078	0.008
	Penalty k=-50	2	0.080	0.008
	Penalty k=-25	2	0.081	0.008
	Penalty k=-25	2	0.078	0.008
MPE	Speaker-Listener	2	0.852	0.085
	Spread	3	4.395	0.439
	Adversary	3	1.651	0.165
	Tag	4	4.911	0.491
SMAC	2s_vs_1sc	2	7.798	0.780
	3s5z	8	18.656	1.866
	MMM2	10	22.480	2.248
	corridor	6	24.890	2.489
	3s_vs_5z	3	10.266	1.027
LBF	15x15-4p-3f	4	2.516	0.252
	8x8-2p-2f-2s-c	2	1.210	0.121
	10x10-3p-3f-2s	3	1.662	0.166
	8x8-2p-2f-c	2	1.253	0.125
	15x15-4p-5f	4	2.559	0.256
	15x15-3p-5f	3	1.939	0.194
	10x10-3p-3f	3	1.849	0.185
RWARE	Tiny 2p	2	4.469	0.447
	Tiny 4p	4	7.976	0.798
	Small 4p	4	7.974	0.797

While matrix games are unsurprisingly the fastest environments to simulate due to their simplicity, Level-Based Foraging tasks are faster to simulate compared to all other environments aside simplest Multi-Agent Particle environment tasks. Despite their arguably more complex environments, the Multi-Robot Warehouse environment is only marginally more expensive to simulate compared to the Multi-Agent Particle environment. It is also worth noting that the cost of simulation of Level-Based Foraging and Multi-Robot Warehouse tasks mostly depends on the number of agents. This allows to simulate large warehouses without additional cost. Unsurprisingly, the Starcraft Multi-Agent Challenge is by far the most expensive environment to simulate as it requires to run the complex game of StarCraft II.

Speed comparisons are conducted on a personal computer running Ubuntu 20.04.2 with a Intel Core i7-10750H CPU (six cores at 2.60GHz) and 16GB of RAM. Simulation was executed on a single CPU thread.

B The EPyMARL Codebase

As part of this work we extended the well-known PyMARL codebase [Samvelyan et al., 2019] to include more algorithms, support more environments as well as allow for more flexible tuning of the implementation details.

B.1 Motivation

Implementation details in Deep RL algorithms significantly affect their achieved returns [Engstrom et al., 2019]. This problem becomes even more apparent in deep MARL research, where the existence of multiple agents significantly increases the amount of implementation details that affect the performance. Therefore, it is often difficult to measure the benefits of newly proposed algorithms compared to existing ones. We believe that a unified codebase, which implements the majority of MARL algorithms used as building blocks in research, would significantly benefit the community. Finally, EPyMARL allows for tuning of additional implementation details compared to PyMARL, including parameter sharing, reward standardisation and entropy regularisation.

B.2 Accessibility and Licensing

All code for EPyMARL is publicly available open-source on GitHub under the following link: `https://github.com/uoe-agents/epymarl`.

All source code that has been taken from the PyMARL repository was licensed (and remains so) under the Apache License v2.0 (included in LICENSE file). Any new code is also licensed under the Apache License v2.0. The NOTICE file in the GitHub repository contains information about the files that have been added or modified compared to the original PyMARL codebase.

B.3 Installation

In the GitHub repository of EPyMARL, the file *requirements.txt* contains all Python packages that have to be install as dependencies in order to use the codebase. We recommend using a Python virtual environment for training MARL algorithm using EPyMARL. After activating the virtual environment, the following commands will install the required packages

```
$ git clone git@github.com:uoe-agents/epymarl.git
$ cd epymarl
$ pip install -r requirements.txt
```

B.4 Execution

EPyMARL is written in Python 3. The neural networks and their operations are implemented using the Pytorch framework [Paszke et al., 2019]. To train an algorithm (QMIX in this example) in a Gym-based environment, execute the following command from the home folder of EPyMARL:

```
$ python3 src/main.py --config=qmix --env-config=gymma with
  ↪  env_args.time_limit=50 env_args.key="lbforaging:Foraging-8x8-2p-3f-v0"
```

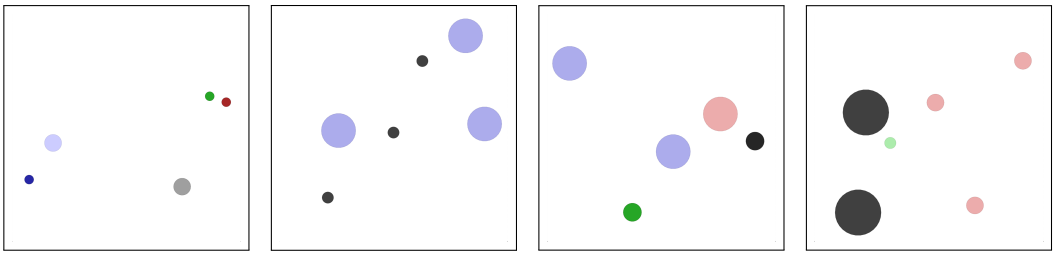
where `config` is the configuration of the algorithm, `gymma` is a Gym compatible wrapper, `env_args.time_limit` is the time limit of the task (number of steps before the episode ends), and `env_args.key` is the name of the task. Default configuration for all algorithms can be found in the `src/config/algs` folder of the codebase.

C Task Specifications

Below, we provide details and descriptions of the environments and tasks used for the evaluation.

C.1 Multi-Agent Particle Environment [Mordatch and Abbeel, 2017]

This environment consists of multiple tasks involving the cooperation and competition between agents. All tasks involve particles and landmarks in a continuous two-dimensional environment. Observations consist of high-level feature vectors and agents are receiving dense reward signals. The action space among all tasks and agents is discrete and usually includes five possible actions corresponding to no movement, move right, move left, move up or move down. All experiments in this environment are executed with a maximum episode length of 25, i.e. episodes are terminated after 25 steps and a new episode is started. All considered tasks are visualised in Figure 6.



(a) (b) (c) (d)

Figure 6: Illustration of MPE tasks (a) Speaker-Listener, (b) Spread, (c) Adversary and (d) Predator-Prey.

MPE Speaker-Listener: In this task, one static speaker agent has to communicate a goal landmark to a listening agent capable of moving. There are a total of three landmarks in the environment and both agents are rewarded with the negative Euclidean distance of the listener agent towards the goal landmark. The speaker agent only observes the colour of the goal landmark. Meanwhile, the listener agent receives its velocity, relative position to each landmark and the communication of the speaker agent as its observation. As actions, the speaker agent has three possible options which have to be trained to encode the goal landmark while the listener agent follows the typical five discrete movement actions of MPE tasks.

MPE Spread: In this task, three agents are trained to move to three landmarks while avoiding collisions with each other. All agents receive their velocity, position, relative position to all other agents and landmarks. The action space of each agent contains five discrete movement actions. Agents are rewarded with the sum of negative minimum distances from each landmark to any agent and a additional term is added to punish collisions among agents.

MPE Adversary: In this task, two cooperating agents compete with a third adversary agent. There are two landmarks out of which one is randomly selected to be the goal landmark. Cooperative agents receive their relative position to the goal as well as relative position to all other agents and landmarks as observations. However, the adversary agent observes all relative positions without receiving information about the goal landmark. All agents have five discrete movement actions. Agents are rewarded with the negative minimum distance to the goal while the cooperative agents are additionally rewarded for the distance of the adversary agent to the goal landmark. Therefore, the cooperative agents have to move to both landmarks to avoid the adversary from identifying which landmark is the goal and reaching it as well. For this competitive scenario, we use a fully cooperative version where the adversary agent is controlled by a pretrained model obtained by training all agents using the MADDPG algorithm for 25,000 episodes.

MPE Predator-Prey: In this task, three cooperating predators hunt a forth agent controlling a faster prey. Two landmarks are placed in the environment as obstacles. All agents receive their own velocity and position as well as relative positions to all other landmarks and agents as observations. Predator agents also observe the velocity of the prey. All agents choose among five movement actions. The agent controlling the prey is punished for any collisions with predators as well as for leaving the observable environment area (to prevent it from simply running away without needing to learn to evade). Predator agents are collectively rewarded for collisions with the prey. We employ a fully cooperative version of this task with a pretrained prey agent. Just as for the Adversary task, the model for the prey is obtained by training all agents using the MADDPG algorithm for 25,000 episodes.

C.2 StarCraft Multi-Agent Challenge [Samvelyan et al., 2019]

The StarCraft Multi-Agent Challenge is a set of fully cooperative, partially observable multi-agent tasks. This environment implements a variety of micromanagement tasks based on the popular real-time strategy game StarCraft II[5] and makes use of the StarCraft II Learning Environment (SC2LE) [Vinyals et al., 2017]. Each task is a specific combat scenario in which a team of agents, each agent controlling an individual unit, battles against an army controlled by the centralised built-in AI of

[5]StarCraft II is a trademark of Blizzard Entertainment™.

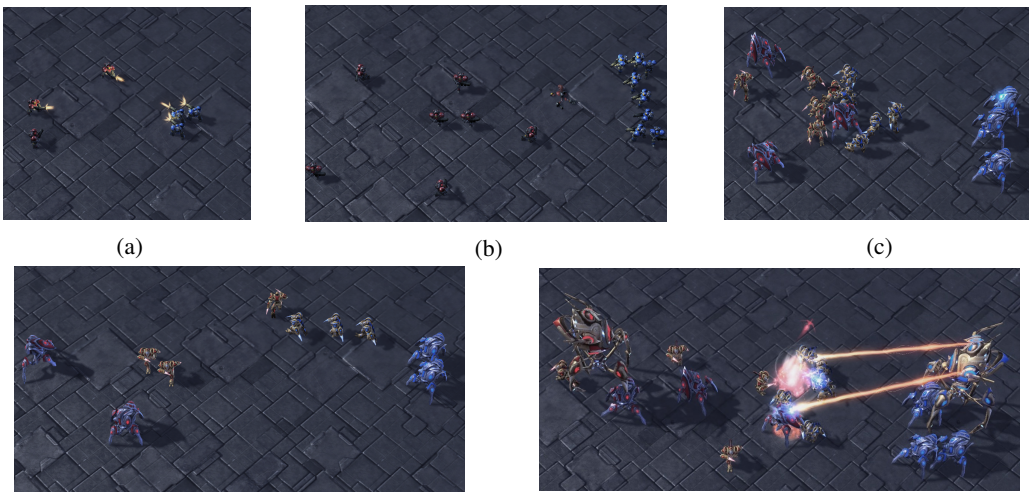


Figure 7: Examples of SMAC tasks with various team configurations and unit types.

the StarCraft game. These tasks require agents to learn precise sequences of actions to enable skills like *kiting* as well as coordinate their actions to focus their attention on specific opposing units. All considered tasks are symmetric in their structure, i.e. both armies consist of the same units. Figure 7 visualises each considered task in this environment.

SMAC 2s_vs_1sc: In this scenario, agents control two stalker units and defeat the enemy team consisting of a single, game-controlled spine crawler.

SMAC 3s5z: In this symmetric scenario, each team controls three stalkers and five zerglings for a total of eight agents.

SMAC MMM2: In this symmetric scenario, each team controls seven marines, two marauders, and one medivac unit. The medivac unit assists other team members by healing them instead of inflicting damage to the enemy team.

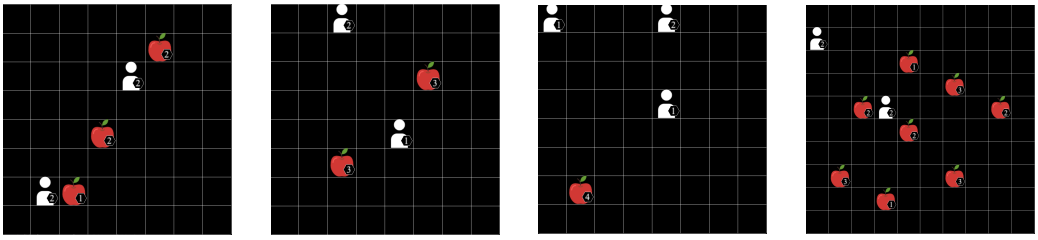
SMAC corridor: In this asymmetric scenario, agents control six zealots fighting an enemy team of 24 zerglings controlled by the game. This tasks requires agents to make effective use of terrain features and employ certain game-specific tricks to win.

SMAC 3s_vs_5z: Finally, in this scenario a team of three stalkers is controlled by agents to fight against a team of five game-controlled zerglings.

C.3 Level-Based Foraging [Albrecht and Ramamoorthy, 2013]

The Level-Based Foraging environment consists of tasks focusing on the coordination of involved agents. The task for each agent is to navigate the grid-world map and collect items. Each agent and item is assigned a level and items are randomly scattered in the environment. In order to collect an item, agents have to choose a certain action next to the item. However, such collection is only successful if the sum of involved agents' levels is equal or greater than the item level. Agents receive reward equal to the level of the collected item. Figure 8 shows the tasks used for our experiments. Unless otherwise specified, every agent can observe the whole map, including the positions and levels of all the entities and can choose to act by moving in one of four directions or attempt to pick up an item.

The tasks that were selected are denoted first by the grid-world size (e.g. 15×15 means a 15 by 15 grid-world). Then, the number of agents is shown (e.g. "4p" means four agents/players), and the number of items scattered in the grid (e.g. "3f" means three food items). We also have some special flags, the "2s" which denotes partial observability with a range of two squares. In those tasks, agents can only observe items or other agents as long as they are located in a square of size 5x5 centred



(a) (b) (c) (d)

Figure 8: Examples of LBF tasks with variable agents, grid-sizes and food items.

around them. Finally, the flag "c" means a cooperative-only variant were all items can only be picked up if all agents in the level attempt to load it simultaneously.

There are many aspects of LBF that are interesting. An increased number of agents and grid-world size naturally makes the problem harder, requiring agents to coordinate to cover a larger area. In addition, partial observability tasks could benefit from agents modelling other agents since it can improve coordination. A tasks that we were unable to get any algorithm to learn is the cooperative-only variant with three or more agents. In that case, many agents are required to coordinate to be able to gather any reward, making the task very hard to solve given the sparsity of the rewards. This last task could on its own provide an interesting challenge for research on multi-agent intrinsic exploration.

Notably, LBF is not only designed for a cooperative reward. In other settings, it can work as a mixed competitive/cooperative environment, where agents must switch between cooperating (for gathering items that they can only pick with others) and competing for items that they can load on their own (without sharing the item reward).

C.4 Multi-Robot Warehouse

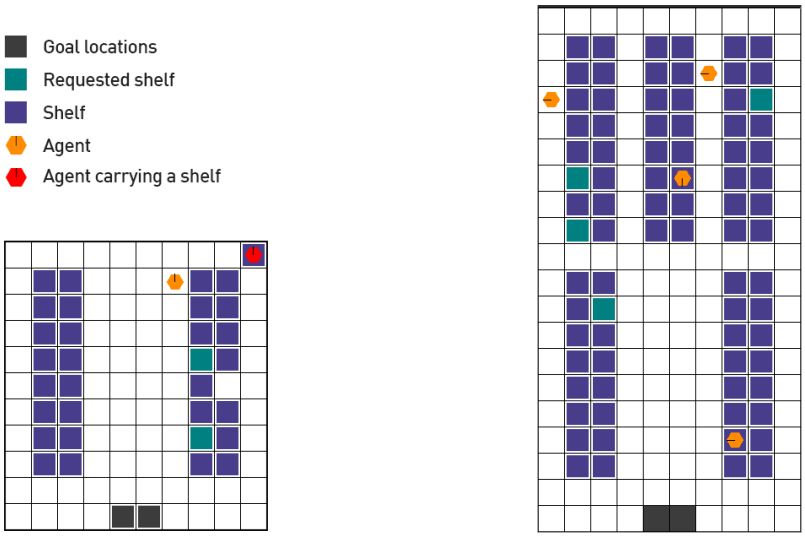
The multi-robot warehouse environment is a set of collaborative, partially observable multi-agent tasks simulating a warehouse operated by robots. Each agent controls a single robot aiming to collect requested shelves. At all times, N shelves are requested and each timestep a request is delivered to the goal location, a new (currently unrequested) shelf is uniformly sampled and added to the list of requests. Agents observe a 3×3 grid including information about potentially close agents, given by their location and rotation, as well as information on surrounding shelves and a list of requests. The action space is discrete and contains of four actions corresponding to turning left or right, moving forward and loading or unloading a shelf. Agents are only rewarded whenever an agent is delivering a requested shelf to a goal position. Therefore, a very specific and long sequence of actions is required to receive any non-zero rewards, making this environment very sparsely rewarded. We use multi-robot warehouse tasks with warehouses of varying size and number of agents N (which is also equal to the number of requested shelves). Figure 9 illustrates the tiny and small warehouses with 2 agents.

For this environment, we have defined three tasks. The "tiny" map is a grid-world of 11 by 11 squares and the "'small" is 11 by 20 squares. The "2p" and "4p" signify two and four robots (agents) respectively. The larger map is considerably harder given the increased sparsity of the rewards.

There are many challenges in the multi-robot warehouse domain as well. The tasks we test in the benchmark paper are limited because of the algorithms' inability to learn and decompose the reward under such sparse reward settings. However, if this challenge is surpassed, either by using a non-cooperative setting or a new algorithm, then the domain offers additional challenges by requiring algorithms to scale to a large number of agents, communicate intentions to increase efficiency and more. This environment is based on a real-world problem, and scaling to scenarios with large maps and hundreds of agents still requires a considerable research effort.

D Computational Cost

Approximately 138,916 CPU hours were spent for executing the experiments presented in the paper without considering the CPU hours required for the hyperparameter search. Figure 10 presents the



(a) (b)

Figure 9: Illustrations of (a) Tiny 2p and (b) Small 4p.

cumulative CPU hours required to train each algorithm in each environments (summed over the different tasks and seeds) with and without parameter sharing using the best identified hyperparameter configurations reported in Appendix I. We observe that the computational cost of running experiments in SMAC is significantly higher compared to any other environment. Finally, the CPU hours required for training the algorithms without sharing is slightly higher compared to training with parameter sharing.

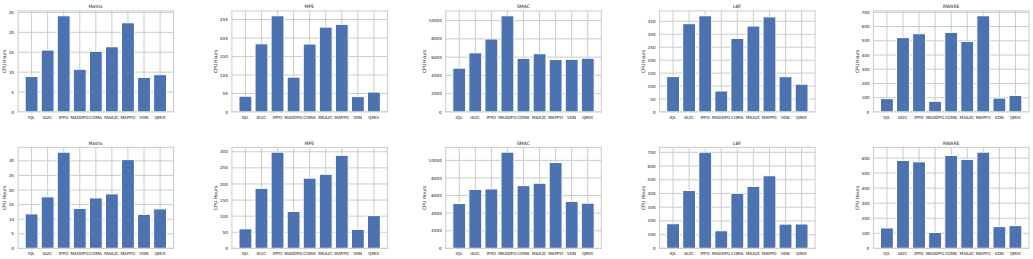


Figure 10: CPU hours required to execute the experiments for each algorithm and environment with (top row) and without (bottom row) parameter sharing.

E Additional Results

Table 6 presents the average returns over training of the nine algorithms in the 25 different tasks with parameter sharing. Tables 7 and 8 present the maximum and the average returns respectively of the nine algorithms in the 25 different tasks without parameter sharing.

Table 6: Average returns and 95% confidence interval over five seeds for all nine algorithms with parameter sharing in all 25 tasks.

	Tasks \Algs.	IQL	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	134.65 ± 0.63	169.34 ± 1.09	170.70 ± 1.77	156.45 ± 8.09	177.31 ± 49.52	167.89 ± 4.36	170.76 ± 1.79	125.50 ± 0.54	125.50 ± 0.54
	Penalty k=0	245.64 ± 1.70	244.27 ± 1.13	247.44 ± 0.59	246.39 ± 0.45	245.63 ± 0.64	244.78 ± 0.87	247.69 ± 0.09	239.80 ± 2.93	243.76 ± 2.36
	Penalty k=-25	44.65 ± 2.67	48.29 ± 0.30	48.44 ± 0.21	48.59 ± 0.05	48.33 ± 0.20	48.45 ± 0.12	48.46 ± 0.22	43.32 ± 5.98	45.99 ± 3.27
	Penalty k=-50	39.70 ± 5.15	46.56 ± 0.57	46.79 ± 0.48	47.22 ± 0.17	46.61 ± 0.44	46.81 ± 0.28	46.82 ± 0.49	39.70 ± 9.63	42.28 ± 6.31
	Penalty k=-75	34.75 ± 7.62	44.82 ± 0.84	45.15 ± 0.75	45.83 ± 0.28	44.88 ± 0.68	45.16 ± 0.43	45.19 ± 0.76	34.75 ± 14.26	38.56 ± 9.34
	Penalty k=-100	29.80 ± 10.10	43.08 ± 1.13	43.50 ± 1.02	44.42 ± 0.36	43.15 ± 0.93	43.51 ± 0.59	43.55 ± 1.04	29.80 ± 18.89	34.85 ± 12.37
MPE	Speaker-Listener	-27.64 ± 3.90	-17.61 ± 2.99	-17.42 ± 3.23	-18.46 ± 0.68	-38.20 ± 6.29	-15.17 ± 0.44	-15.01 ± 0.64	-27.41 ± 3.11	-21.29 ± 2.79
	Spread	-155.81 ± 1.50	-152.72 ± 0.96	-149.89 ± 2.91	-157.10 ± 2.30	-245.22 ± 84.46	-144.73 ± 4.09	-149.26 ± 0.94	-148.57 ± 1.67	-154.70 ± 4.90
	Adversary	7.58 ± 0.14	10.18 ± 0.05	10.21 ± 0.16	7.80 ± 1.43	6.12 ± 0.35	10.11 ± 0.14	9.61 ± 0.07	7.64 ± 0.21	8.11 ± 0.37
	Tag	13.70 ± 1.97	12.43 ± 1.05	13.60 ± 2.95	6.65 ± 3.90	5.11 ± 0.58	11.93 ± 2.09	13.78 ± 4.40	15.24 ± 1.59	15.00 ± 2.73
SMAC	2s_vs_1sc	14.76 ± 0.45	19.74 ± 0.02	19.44 ± 0.29	10.15 ± 1.32	9.04 ± 0.83	17.89 ± 0.85	19.67 ± 0.09	16.11 ± 0.23	15.98 ± 0.77
	3s5z	14.09 ± 0.28	14.84 ± 1.29	11.80 ± 1.51	8.60 ± 2.35	15.51 ± 0.98	18.82 ± 0.14	19.09 ± 0.38	17.85 ± 0.25	18.36 ± 0.07
	corridor	10.91 ± 0.82	13.14 ± 1.24	14.60 ± 3.43	5.15 ± 0.25	7.00 ± 0.15	7.89 ± 0.28	13.20 ± 2.98	11.14 ± 1.66	11.67 ± 1.88
	MMM2	10.11 ± 0.32	7.31 ± 1.89	9.97 ± 1.33	3.42 ± 0.05	6.50 ± 0.17	9.07 ± 1.35	15.39 ± 0.16	15.93 ± 0.23	15.63 ± 0.32
	3s_vs_5z	17.35 ± 0.23	4.32 ± 0.04	13.38 ± 4.36	5.34 ± 0.47	1.15 ± 1.35	6.17 ± 0.39	13.09 ± 2.63	14.72 ± 4.01	9.68 ± 1.87
LBF	8x8-2p-2f-c	0.75 ± 0.04	0.97 ± 0.00	0.94 ± 0.02	0.32 ± 0.02	0.32 ± 0.12	0.97 ± 0.00	0.95 ± 0.01	0.64 ± 0.09	0.39 ± 0.10
	8x8-2p-2f-2s-c	0.86 ± 0.01	0.97 ± 0.00	0.50 ± 0.01	0.54 ± 0.05	0.24 ± 0.08	0.97 ± 0.00	0.77 ± 0.02	0.83 ± 0.01	0.77 ± 0.03
	10x10-3p-3f	0.54 ± 0.02	0.95 ± 0.01	0.90 ± 0.02	0.20 ± 0.06	0.15 ± 0.05	0.95 ± 0.01	0.91 ± 0.01	0.40 ± 0.05	0.32 ± 0.07
	10x10-3p-3f-2s	0.69 ± 0.02	0.84 ± 0.01	0.62 ± 0.01	0.27 ± 0.02	0.23 ± 0.06	0.85 ± 0.02	0.66 ± 0.01	0.64 ± 0.02	0.67 ± 0.01
	15x15-3p-5f	0.09 ± 0.02	0.61 ± 0.06	0.41 ± 0.09	0.08 ± 0.00	0.06 ± 0.03	0.59 ± 0.09	0.43 ± 0.09	0.08 ± 0.01	0.04 ± 0.01
	15x15-4p-3f	0.24 ± 0.05	0.89 ± 0.03	0.82 ± 0.06	0.13 ± 0.01	0.12 ± 0.03	0.92 ± 0.01	0.79 ± 0.03	0.16 ± 0.03	0.08 ± 0.01
	15x15-4p-5f	0.15 ± 0.03	0.59 ± 0.06	0.40 ± 0.13	0.13 ± 0.01	0.07 ± 0.02	0.73 ± 0.02	0.39 ± 0.14	0.15 ± 0.02	0.09 ± 0.02
RWARE	Tiny 2p	0.04 ± 0.03	2.91 ± 0.45	12.63 ± 1.38	0.11 ± 0.07	0.13 ± 0.05	3.20 ± 0.41	15.42 ± 1.20	0.03 ± 0.01	0.03 ± 0.03
	Tiny 4p	0.33 ± 0.13	10.30 ± 0.93	22.68 ± 7.40	0.28 ± 0.03	0.39 ± 0.06	14.39 ± 4.01	40.17 ± 1.42	0.29 ± 0.13	0.10 ± 0.09
	Small 4p	0.03 ± 0.04	2.45 ± 0.18	9.19 ± 2.36	0.06 ± 0.02	0.08 ± 0.01	3.48 ± 0.42	18.12 ± 1.11	0.02 ± 0.03	0.01 ± 0.01

Table 7: Maximum returns and 95% confidence interval over five seeds for all nine algorithms without parameter sharing in all 25 tasks.

	Tasks \Algs.	IQL	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	150.00 ± 0.00	175.00 ± 0.00	175.00 ± 0.00	170.00 ± 10.00	195.00 ± 40.00	175.00 ± 0.00	175.00 ± 0.00	150.00 ± 0.00	155.00 ± 10.00
	Penalty k=0	250.00 ± 0.00	250.00 ± 0.00	250.00 ± 0.00	249.94 ± 0.08	250.00 ± 0.00	250.00 ± 0.00	250.00 ± 0.00	250.00 ± 0.00	250.00 ± 0.00
	Penalty k=-25	50.00 ± 0.00	50.00 ± 0.00	90.00 ± 80.00	89.99 ± 80.01	50.00 ± 0.00	130.00 ± 97.98	90.00 ± 80.00	50.00 ± 100.00	50.00 ± 100.00
	Penalty k=-50	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00	49.98 ± 0.01	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 100.00	50.00 ± 100.00
	Penalty k=-75	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00	49.98 ± 0.01	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 100.00	50.00 ± 100.00
	Penalty k=-100	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00	49.98 ± 0.01	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 100.00	50.00 ± 100.00
MPE	Speaker-Listener	-18.61 ± 5.65	-17.08 ± 3.45	-15.56 ± 4.40 *	-12.73 ± 0.73 *	-26.50 ± 0.50	-13.66 ± 3.67 *	-14.35 ± 3.56 *	-15.47 ± 1.26	$\mathbf{-11.59 \pm 0.67}$
	Spread	-141.87 ± 1.68	-131.74 ± 4.33 *	-132.46 ± 3.54 *	-136.73 ± 0.83	-169.04 ± 2.72	-130.88 ± 2.44 *	$\mathbf{-128.64 \pm 2.83}$	-142.13 ± 1.86	-130.97 ± 2.51 *
	Adversary	9.09 ± 0.52	10.80 ± 1.97 *	11.17 ± 0.85 *	8.81 ± 0.61	9.18 ± 0.43	10.88 ± 2.43 *	$\mathbf{12.04 \pm 0.53}$	9.34 ± 0.57	11.32 ± 0.78 *
	Tag	19.18 ± 2.30	16.04 ± 8.08 *	18.46 ± 5.19 *	2.82 ± 3.56	19.14 ± 7.50 *	26.50 ± 1.42 *	17.96 ± 8.82 *	18.44 ± 2.51	$\mathbf{26.88 \pm 5.61}$
SMAC	2s_vs_1sc	15.73 ± 1.08	20.23 ± 0.01	20.15 ± 0.10 *	10.35 ± 2.20	18.48 ± 3.28 *	19.88 ± 0.38 *	$\mathbf{20.25 \pm 0.01}$	17.22 ± 0.90	18.83 ± 0.47
	3s5z	16.85 ± 1.43	14.44 ± 2.08	14.77 ± 2.51	15.05 ± 0.81	19.55 ± 0.45 *	19.38 ± 0.30	$\mathbf{19.77 \pm 0.11}$	19.08 ± 0.29	18.40 ± 0.70
	MMM2	10.86 ± 1.04	8.38 ± 2.90	7.35 ± 0.48	4.92 ± 0.10	4.98 ± 0.42	10.79 ± 0.34	10.02 ± 0.19	$\mathbf{16.20 \pm 0.44}$	12.27 ± 2.28
	corridor	11.06 ± 1.36	13.11 ± 4.27 *	10.29 ± 4.60	6.57 ± 0.53	8.34 ± 0.81	9.26 ± 3.08	8.51 ± 0.76	11.42 ± 2.18 *	$\mathbf{15.12 \pm 3.42}$
	3s_vs_5z	11.80 ± 1.05 *	4.41 ± 0.02	4.26 ± 0.24	6.21 ± 1.97	4.13 ± 0.05	5.39 ± 0.74	4.62 ± 0.11	14.42 ± 2.22 *	$\mathbf{15.13 \pm 3.86}$
LBF	8x8-2p-2f-c	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.43 ± 0.02	0.95 ± 0.07 *	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.99 ± 0.01 *	0.57 ± 0.15
	8x8-2p-2f-2s-c	0.99 ± 0.01 *	$\mathbf{1.00 \pm 0.00}$	0.83 ± 0.03	0.66 ± 0.04	0.92 ± 0.02	$\mathbf{1.00 \pm 0.00}$	0.92 ± 0.05	0.99 ± 0.01 *	0.98 ± 0.01
	10x10-3p-3f	0.54 ± 0.12	$\mathbf{1.00 \pm 0.01}$	0.96 ± 0.01	0.20 ± 0.03	0.31 ± 0.08	0.99 ± 0.01 *	0.98 ± 0.01	0.39 ± 0.06	0.22 ± 0.03
	10x10-3p-3f-2s	0.77 ± 0.06	0.89 ± 0.02	0.72 ± 0.02	0.29 ± 0.02	0.27 ± 0.09	$\mathbf{0.96 \pm 0.01}$	0.70 ± 0.04	0.76 ± 0.04	0.78 ± 0.05
	15x15-3p-5f	0.11 ± 0.02	$\mathbf{0.74 \pm 0.12}$	0.62 ± 0.12 *	0.10 ± 0.01	0.09 ± 0.02	0.72 ± 0.05 *	0.49 ± 0.19 *	0.11 ± 0.01	0.06 ± 0.01
	15x15-4p-3f	0.16 ± 0.03	$\mathbf{0.99 \pm 0.00}$	0.90 ± 0.01	0.17 ± 0.01	0.21 ± 0.07	0.94 ± 0.05 *	0.89 ± 0.02	0.13 ± 0.02	0.10 ± 0.01
	15x15-4p-5f	0.17 ± 0.00	$\mathbf{0.77 \pm 0.10}$	0.69 ± 0.05 *	0.15 ± 0.01	0.14 ± 0.03	0.76 ± 0.04 *	0.45 ± 0.17	0.14 ± 0.01	0.08 ± 0.01
RWARE	Tiny 2p	0.06 ± 0.08	5.56 ± 1.68	8.70 ± 4.04	0.24 ± 0.21	1.46 ± 0.34	6.16 ± 1.94	$\mathbf{17.48 \pm 4.52}$	0.06 ± 0.08	0.06 ± 0.08
	Tiny 4p	0.44 ± 0.10	18.02 ± 5.87	14.10 ± 5.20	0.44 ± 0.24	1.48 ± 0.48	31.56 ± 4.29	$\mathbf{38.74 \pm 2.99}$	0.34 ± 0.17	0.16 ± 0.15
	Small 4p	0.04 ± 0.05	3.10 ± 0.68	5.78 ± 0.42	0.16 ± 0.12	0.16 ± 0.10	5.00 ± 0.68	$\mathbf{13.78 \pm 1.63}$	0.04 ± 0.05	0.06 ± 0.08

Table 8: Average returns and 95% confidence interval over five seeds for all nine algorithms without parameter sharing in all 25 tasks.

	Tasks \Algs.	IQL	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
Matrix Games	Climbing	130.20 ± 4.37	164.72 ± 0.69	171.68 ± 0.41	150.05 ± 3.75	187.50 ± 41.19	169.43 ± 1.01	171.62 ± 0.39	132.52 ± 3.20	132.43 ± 3.37
	Penalty k=0	246.73 ± 1.39	243.65 ± 0.93	247.72 ± 0.04	247.88 ± 0.05	247.10 ± 0.38	246.61 ± 0.52	247.73 ± 0.03	247.03 ± 1.85	247.03 ± 0.99
	Penalty k=-25	49.70 ± 0.24	46.95 ± 0.16	88.05 ± 79.01	86.14 ± 75.19	48.32 ± 0.10	125.90 ± 95.14	88.06 ± 79.03	50.00 ± 0.99	50.00 ± 0.99
	Penalty k=-50	49.70 ± 0.24	44.31 ± 0.34	46.99 ± 0.23	47.12 ± 0.07	46.54 ± 0.20	46.56 ± 0.33	46.99 ± 0.24	50.00 ± 0.99	50.00 ± 0.99
	Penalty k=-76	49.70 ± 0.24	41.64 ± 0.50	45.44 ± 0.36	45.74 ± 0.12	44.77 ± 0.31	44.88 ± 0.47	45.44 ± 0.37	50.00 ± 0.99	50.00 ± 0.99
	Penalty k=-100	49.70 ± 0.24	38.94 ± 0.66	43.89 ± 0.48	44.33 ± 0.16	42.99 ± 0.41	43.21 ± 0.62	43.90 ± 0.52	50.00 ± 0.99	50.00 ± 0.99
MPE	Speaker-Listener	-30.49 ± 4.67	-23.33 ± 2.67	-22.78 ± 3.10	-17.79 ± 0.97	-33.88 ± 3.38	-19.48 ± 2.92	-20.51 ± 2.85	-29.49 ± 2.36	-20.31 ± 1.84
	Spread	-160.10 ± 1.59	-141.31 ± 3.86	-142.86 ± 4.49	-149.53 ± 2.41	-184.72 ± 2.99	-139.54 ± 4.78	-139.20 ± 4.58	-158.60 ± 2.06	-157.04 ± 1.38
	Adversary	7.82 ± 0.19	9.18 ± 1.52	9.46 ± 1.02	7.24 ± 2.13	7.28 ± 0.76	9.43 ± 1.93	10.20 ± 0.24	8.06 ± 0.27	8.81 ± 0.52
	Tag	12.59 ± 1.60	9.59 ± 4.30	11.90 ± 3.31	1.91 ± 2.09	14.28 ± 5.43	11.79 ± 5.40	10.90 ± 5.47	10.71 ± 0.69	14.27 ± 2.81
SMAC	2s_vs_1sc	13.37 ± 0.35	19.69 ± 0.05	8.59 ± 1.90	7.91 ± 0.16	15.32 ± 3.21	16.22 ± 3.75	19.68 ± 0.08	15.12 ± 0.46	15.66 ± 0.82
	3s5z	14.23 ± 0.84	12.65 ± 1.00	12.26 ± 0.98	7.65 ± 0.54	17.13 ± 1.11	17.94 ± 0.28	19.03 ± 0.19	17.06 ± 0.28	15.46 ± 0.59
	MMM2	8.27 ± 0.64	6.98 ± 1.72	6.23 ± 0.88	3.11 ± 0.12	3.82 ± 0.36	9.85 ± 0.19	9.41 ± 0.19	12.72 ± 0.56	8.05 ± 2.05
	corridor	8.38 ± 0.82	9.81 ± 1.98	7.80 ± 0.68	4.99 ± 0.17	7.85 ± 0.46	7.78 ± 0.73	8.18 ± 0.43	9.25 ± 0.85	10.76 ± 1.79
	3s_vs_5z	9.15 ± 0.46	4.26 ± 0.02	4.20 ± 0.22	3.53 ± 0.39	3.20 ± 0.89	4.91 ± 0.41	4.55 ± 0.02	10.85 ± 1.61	10.09 ± 1.10
LBF	8x8-2p-2f-c	0.95 ± 0.02	0.96 ± 0.01	0.91 ± 0.02	0.40 ± 0.05	0.63 ± 0.13	0.97 ± 0.00	0.93 ± 0.02	0.93 ± 0.00	0.90 ± 0.00
	8x8-2p-2f-2s-c	0.97 ± 0.00	0.96 ± 0.00	0.50 ± 0.00	0.52 ± 0.02	0.63 ± 0.08	0.97 ± 0.00	0.79 ± 0.03	0.96 ± 0.00	0.96 ± 0.00
	10x10-3p-3f	0.77 ± 0.05	0.92 ± 0.01	0.85 ± 0.01	0.15 ± 0.00	0.24 ± 0.03	0.92 ± 0.01	0.85 ± 0.02	0.80 ± 0.03	0.81 ± 0.04
	10x10-3p-3f-2s	0.78 ± 0.02	0.76 ± 0.01	0.61 ± 0.01	0.22 ± 0.03	0.23 ± 0.05	0.80 ± 0.00	0.62 ± 0.00	0.86 ± 0.02	0.86 ± 0.02
	15x15-3p-5f	0.11 ± 0.02	0.39 ± 0.09	0.33 ± 0.13	0.07 ± 0.00	0.06 ± 0.00	0.40 ± 0.03	0.24 ± 0.09	0.18 ± 0.01	0.08 ± 0.03
	15x15-4p-3f	0.29 ± 0.04	0.81 ± 0.01	0.60 ± 0.01	0.10 ± 0.01	0.16 ± 0.03	0.74 ± 0.03	0.65 ± 0.06	0.55 ± 0.06	0.12 ± 0.04
	15x15-4p-5f	0.17 ± 0.03	0.49 ± 0.10	0.40 ± 0.11	0.11 ± 0.00	0.09 ± 0.01	0.44 ± 0.03	0.25 ± 0.04	0.25 ± 0.01	0.11 ± 0.02
RWARE	Tiny 2p	0.01 ± 0.01	2.02 ± 0.68	5.36 ± 3.05	0.07 ± 0.05	0.37 ± 0.04	2.23 ± 0.45	10.39 ± 3.04	0.01 ± 0.01	0.01 ± 0.01
	Tiny 4p	0.13 ± 0.04	6.10 ± 1.38	9.24 ± 4.20	0.25 ± 0.04	0.50 ± 0.12	10.73 ± 1.31	25.14 ± 1.44	0.10 ± 0.02	0.05 ± 0.02
	Small 4p	0.01 ± 0.00	1.02 ± 0.10	3.64 ± 0.38	0.03 ± 0.02	0.07 ± 0.02	1.27 ± 0.09	7.06 ± 0.63	0.01 ± 0.01	0.01 ± 0.01

F Visualisation of the Evaluation Returns During Training

Figure 11 presents the evaluation returns that are achieved during training by the nine algorithms with parameter sharing in the 25 different tasks.

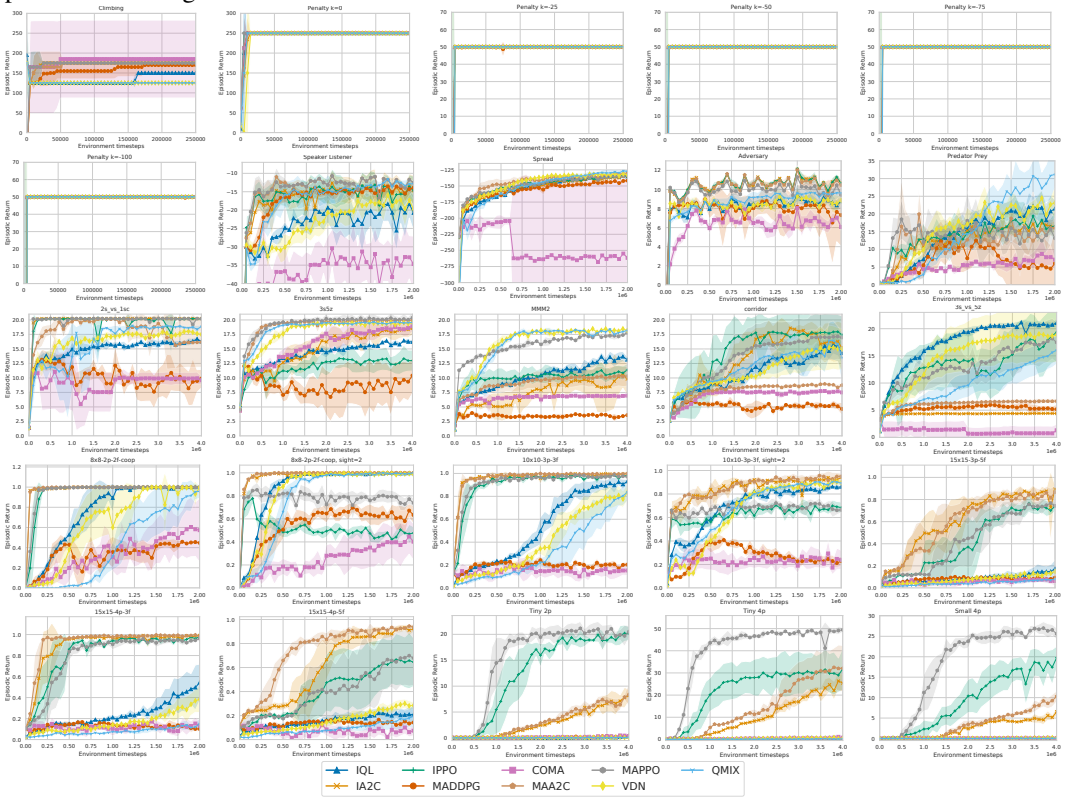


Figure 11: Episodic returns of all algorithms with parameter sharing in all environments showing the mean and the 95% confidence interval over five different seeds.

G SMAC Win-Rates

Table 9: Maximum win-rate and 95% confidence interval over five seeds for all nine algorithms with parameter sharing in all SMAC tasks.

Tasks \Algs.	IQL	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
2s_vs_1sc	0.61 ± 0.04	1.00 ± 0.00	1.00 ± 0.00	0.21 ± 0.20	0.34 ± 0.41	1.00 ± 0.00	1.00 ± 0.00	0.74 ± 0.04	0.85 ± 0.03
3s5z	0.39 ± 0.04	0.72 ± 0.23	0.17 ± 0.13	0.15 ± 0.30	0.81 ± 0.19	0.99 ± 0.01	0.96 ± 0.01	0.92 ± 0.05	0.94 ± 0.01
corridor	0.44 ± 0.20	0.80 ± 0.08	0.82 ± 0.25	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.68 ± 0.34	0.44 ± 0.37	0.53 ± 0.29
MMM2	0.27 ± 0.08	0.14 ± 0.17	0.15 ± 0.15	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.73 ± 0.07	0.89 ± 0.04	0.89 ± 0.04
3s_vs_5z	0.67 ± 0.17	0.00 ± 0.00	0.72 ± 0.43	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.41 ± 0.43	0.62 ± 0.31	0.43 ± 0.37

Table 10: Maximum win-rate and 95% confidence interval over five seeds for all nine algorithms without parameter sharing in all SMAC tasks.

Tasks \Algs.	IQL	IA2C	IPPO	MADDPG	COMA	MAA2C	MAPPO	VDN	QMIX
2s_vs_1sc	0.49 ± 0.12	1.00 ± 0.00	0.99 ± 0.01	0.06 ± 0.10	0.79 ± 0.40	0.96 ± 0.04	1.00 ± 0.00	0.64 ± 0.11	0.84 ± 0.02
3s5z	0.48 ± 0.23	0.27 ± 0.25	0.27 ± 0.25	0.13 ± 0.09	0.91 ± 0.08	0.87 ± 0.07	0.94 ± 0.03	0.81 ± 0.06	0.70 ± 0.11
corridor	0.05 ± 0.06	0.31 ± 0.39	0.17 ± 0.30	0.00 ± 0.00	0.01 ± 0.02	0.06 ± 0.12	0.01 ± 0.01	0.08 ± 0.11	0.40 ± 0.32
MMM2	0.04 ± 0.08	0.06 ± 0.13	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.58 ± 0.04	0.23 ± 0.16
3s_vs_5z	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.11 ± 0.22	0.23 ± 0.24

H Hyperparameter Optimisation

Table 11: Range of hyperparameters that was evaluated in each environment. N/A means that this hyperparameter was not optimised, and that we used one that was either proposed in the original paper or was found to be the best in the rest of the environments. If only one value is presented it means that this hyperparameter was used for all algorithms in this task.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64/128	64/128	64/128	64/128	64/128
learning rate	0.0001/0.0003/0.0005	0.0001/0.0003/0.0005	0.0005	0.0001/0.0003/0.0005	0001/0.0003/0.0005
reward standardisation	True/False	True/False	True/False	True/False	True/False
network type	FC	FC/GRU	FC/GRU	FC/GRU	FC/GRU
evaluation epsilon	0.0/0.05	0.0/0.05	0.0/0.05	0.0/0.05	0.0/0.05
epsilon anneal	50,000/200,000	50,000/200,000	50,000	50,000/200,000	50,000/200,000
target update	200(hard)/0.01(soft)	200(hard)/0.01(soft)	N/A	200(hard)/0.01(soft)	200(hard)/0.01(soft)
entropy coefficient	0.01/0.001	0.01/0.001	N/A	0.01/0.001	0.01/0.001
n-step	5/10	5/10	N/A	5/10	5/10

The parameters of each algorithms are optimised for each environment in one of its tasks and are kept constant for the rest of the tasks within the same environment. Each combination of hyperparameters is evaluated for three different seeds. The combination of hyperparameters that achieved the maximum evaluation, averaged over the three seeds, is used for producing the results presented in this work. Table 11 presents the range of hyperparameters we evaluated in each environment, on the respective applicable algorithms. In general, all algorithms were evaluated in approximately the same number of hyperparameter combination for each environment to ensure consistency. To reduce the computational cost, the hyperparameter search was limited in SMAC compared to the other environments. However, several of the evaluated algorithms have been previously evaluated in SMAC and their best hyperparameters are publicly available in their respective papers.

I Selected Hyperparameters

In this section we present the hyperparameters used in each task. In the off-policy algorithms we use an experience replay to break the correlation between consecutive samples [Lin, 1992, Mnih et al., 2015]. In the on-policy algorithms we use parallel synchronous workers to break the correlation between consecutive samples [Mnih et al., 2015]. The size of the experience replay is either 5K episodes or 1M samples, depending on which is smaller in terms of used memory. Exploration in Q-based algorithms is done with epsilon-greedy, starting with $\epsilon = 1$ and linearly reducing it to 0.05. Additionally, in Q-based algorithms we select action with epsilon-greedy (with a small epsilon value) to ensure that the agents are not stuck. The evaluation epsilon is the hyperparameter that is optimised during the hyperparameter optimisation, with possible values between 0 and 0.05. In the stochastic policy algorithms, we perform exploration by sampling their categorical policy. During execution, in the stochastic policy algorithms, we sample their policy instead of computing the action that maximises the policy. The computation of the temporal difference targets is done using the Double Q-learning [Hasselt, 2010] update rule. In IPPO and MAPPO the number of update epochs per training batch is 4 and the clipping value of the surrogate objective is 0.2.

Tables 12 and 13 present the hyperparameters in all environments for the IQL algorithm with and without parameter sharing respectively.

Table 12: Hyperparameters for IQL with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	128	64
learning rate	0.0003	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	False	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	200,000	50,000	200,000	50,000
target update	200 (hard)	0.01 (soft)	200 (hard)	200 (hard)	0.01 (soft)

Table 13: Hyperparameters for IQL without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	64	64	64
learning rate	0.0001	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	200,000	50,000	50,000	50,000
target update	0.01 (soft)	0.01 (soft)	200 (hard)	200 (hard)	0.01 (soft)

Tables 14 and 15 present the hyperparameters in all environments for the IA2C algorithm with and without parameter sharing respectively.

Table 14: Hyperparameters for IA2C with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	128	64
learning rate	0.0005	0.0005	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.01	0.01	0.01	0.001	0.01
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	5	5	5	5	5

Table 15: Hyperparameters for IA2C without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	64	64	64
learning rate	0.0001	0.0005	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	FC	GRU	FC
entropy coefficient	0.01	0.01	0.01	0.01	0.01
target update	200 (hard)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	5	10	5	5	5

Tables 16 and 17 present the hyperparameters in all environments for the IPPO algorithm with and without parameter sharing respectively.

Table 16: Hyperparameters for IPPO with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	128	128
learning rate	0.0005	0.0003	0.0005	0.0003	0.0005
reward standardisation	True	True	False	False	False
network type	FC	GRU	GRU	FC	GRU
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	5	10	5	10

Table 17: Hyperparameters for IPPO without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	64	128	128
learning rate	0.0005	0.0001	0.0005	0.0001	0.0005
reward standardisation	True	True	True	False	False
network type	FC	FC	FC	GRU	FC
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	10	10	5	10

Tables 18 and 19 present the hyperparameters in all environments for the MADDPG algorithm with and without parameter sharing respectively.

Table 18: Hyperparameters for MADDPG with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	64	64
learning rate	0.0003	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	False	True	False
network type	FC	GRU	GRU	FC	FC
actor regularisation	0.001	0.001	0.01	0.001	0.001
target update	200 (hard)	200 (hard)	0.01 (soft)	200 (hard)	0.01 (soft)

Table 19: Hyperparameters for MADDPG without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	64	64
learning rate	0.0005	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	True	True	False
network type	FC	GRU	FC	FC	FC
actor regularisation	0.001	0.01	0.001	0.001	0.001
target update	200 (hard)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)

Tables 20 and 21 present the hyperparameters in all environments for the COMA algorithm with and without parameter sharing respectively.

Table 20: Hyperparameters for COMA with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	128	64
learning rate	0.0005	0.0003	0.0005	0.0001	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.01	0.001	0.01	0.001	0.01
target update	0.01 (soft)	200 (hard)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	10	5	10	5

Table 21: Hyperparameters for COMA without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	128	64
learning rate	0.0003	0.0005	0.0005	0.0001	0.0005
reward standardisation	True	True	True	True	False
network type	FC	GRU	GRU	GRU	FC
entropy coefficient	0.01	0.01	0.01	0.001	0.01
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	10	10	5	5	5

Tables 22 and 23 present the hyperparameters in all environments for the MAA2C algorithm with and without parameter sharing respectively.

Table 22: Hyperparameters for MAA2C with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	128	128	64
learning rate	0.003	0.0005	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.001	0.01	0.01	0.01	0.01
target update	0.01	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	10	5	5	10	5

Table 23: Hyperparameters for MAA2C without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	128	128	64
learning rate	0.0005	0.0003	0.0005	0.0005	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	FC	GRU	FC
entropy coefficient	0.001	0.01	0.01	0.01	0.01
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	10	5	5	5	5

Tables 24 and 25 present the hyperparameters in all environments for the MAPPO algorithm with and without parameter sharing respectively.

Table 24: Hyperparameters for MAPPO with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	64	128	128
learning rate	0.0005	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	False	False	False
network type	FC	FC	GRU	FC	FC
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)	0.01 (soft)
n-step	5	5	10	5	10

Table 25: Hyperparameters for MAPPO without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	64	128	128
learning rate	0.0005	0.0001	0.0005	0.0001	0.0005
reward standardisation	True	True	True	False	False
network type	FC	FC	GRU	FC	FC
entropy coefficient	0.001	0.01	0.001	0.001	0.001
target update	0.01 (soft)	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)
n-step	5	5	10	10	10

Tables 26 and 27 present the hyperparameters in all environments for the VDN algorithm with and without parameter sharing respectively.

Table 26: Hyperparameters for VDN with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	128	128	128	64
learning rate	0.0001	0.0005	0.0005	0.0003	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.0	0.05
epsilon anneal	200,000	50,000	50,000	200,000	50,000
target update	0.01 (soft)	200 (hard)	200 (hard)	0.01 (soft)	0.01 (soft)

Table 27: Hyperparameters for VDN without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	64	64	64
learning rate	0.0005	0.0005	0.0005	0.0001	0.0005
reward standardisation	True	True	True	True	True
network type	FC	FC	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	50,000	50,000	50,000	50,000
target update	0.01 (soft)	200 (hard)	200 (hard)	200 (hard)	0.01 (soft)

Tables 28 and 29 present the hyperparameters in all environments for the QMIX algorithm with and without parameter sharing respectively.

Table 28: Hyperparameters for QMIX with parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	64	64	128	64	64
learning rate	0.0003	0.0005	0.005	0.0003	0.0005
reward standardisation	True	True	True	True	True
network type	FC	GRU	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	200,000	200,000	50,000	200,000	50,000
target update	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)	0.01 (soft)

Table 29: Hyperparameters for QMIX without parameter sharing.

	Matrix Games	MPE	SMAC	LBF	RWARE
hidden dimension	128	128	64	64	64
learning rate	0.0005	0.0003	0.0005	0.0001	0.0003
reward standardisation	True	True	True	True	True
network type	FC	GRU	GRU	GRU	FC
evaluation epsilon	0.0	0.0	0.05	0.05	0.05
epsilon anneal	50,000	200,000	50,000	50,000	50,000
target update	0.01 (soft)	0.01 (soft)	200 (hard)	0.01 (soft)	0.01 (soft)