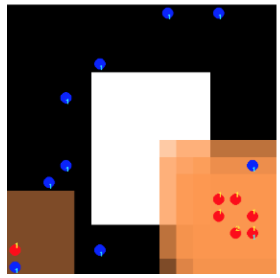

Figure 4: Pursuit Environment

A Algorithm Details

Algorithm 1 SUPER algorithm for DQN

for each training iteration **do**
  Collect a batch of experiences b {DQN}
  for each agent i **do**
    Insert b_i into buffer_i {DQN}
  end for
  for each agent i **do**
    Select $b_i^* \subseteq b_i$ of experiences to share[1] {SUPER}
    for each agent $j \neq i$ **do**
      Insert $b*_i$ into buffer_j {SUPER}
    end for
  end for
  for each agent i **do**
    Sample a train batch b_i from buffer_i {DQN}
    Learn on train batch b_i {DQN}
  end for
end for

[1] See section "Experience Selection"

Algorithm 1 shows a full pseudocode listing for SUPER on DQN.

B Experiment Domains

SISL: Pursuit is a semi-cooperative environment, where a group of pursuers has to capture a group of evaders in a grid-world with an obstacle. The evaders (blue) move randomly, while the pursuers (red) are controlled by RL agents. If a group of two or more agents fully surround an evader, they each receive a reward, and the evader is removed from the environment. The episode ends when all evaders have been captured, or after 500 steps, whichever is earlier. Pursuers also receive a (very small) reward for being adjacent to an evader (even if the evader is not fully surrounded), and a (small) negative reward each timestep, to incentivize them to complete episodes early. We use 8 pursuers and 30 evaders.

MAgent: Battle is a semi-adversarial environment, where two groups of opposing teams are battling against each other. An agent is rewarded 0.2 points for attacking agents in the opposite team, and 5 points if the other agent is killed. All agents start with 10 health points (HP) and lose 2 HP in each attack received, while regaining 0.1 HP in every turn. Once killed, an agent is removed from the environment. An episode ends when all agents from one team are killed. The action space, of size 21 is identical for all agents, with (8) options to attack, (12) to move and one option to do nothing. Since no additional reward is given for collaborating with other agents in the same team, it is considered to be more challenging to form collaboration between agents in this environment. We use a map of size 18×18 and 6 agents per team.

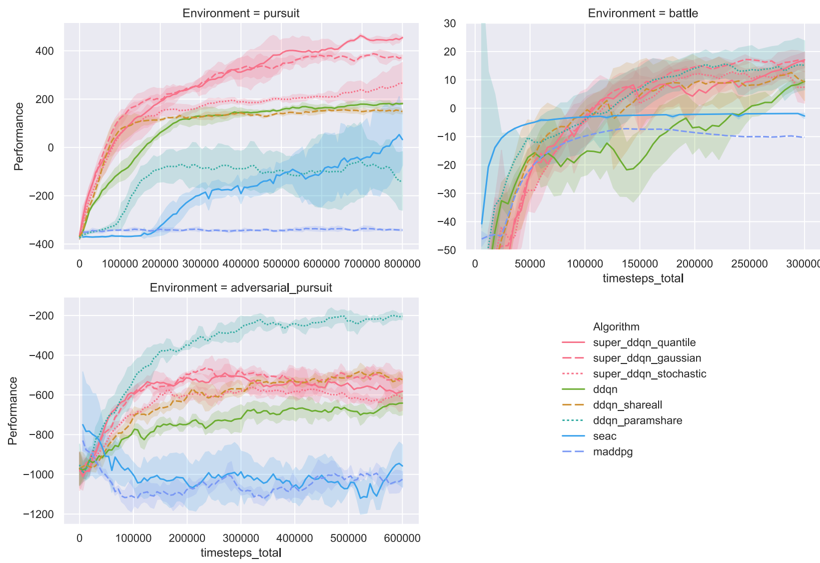

Figure 5: Performance of SUPER-dueling-DDQN variants with target bandwidth 0.1 on all three domains. For Pursuit, performance is the total mean episode reward from all agents. For Battle and Adversarial-Pursuit, performance is the total mean episode reward from all agents in the sharing team (blue team in Battle, prey team in Adversarial-Pursuit). Shaded areas indicate one standard deviation.

MAgent: Adversarial Pursuit is a predator-prey environment, with two types of agents, prey and predator. The predators navigate through obstacles in the map with the purpose of tagging the prey. An agent in the predators team is rewarded 1 point for tagging a prey, while a prey is rewarded -1 when being tagged by a predator. Unlike in the Battle environment, prey agents are not removed from the game when being tagged. Note that prey agents are provided only with a negative or zero reward (when manage to avoid attacks), and their aim is thus to evade predator agents. We use 8 prey agents, and 4 predator agents.

C Further Experimental Results

C.1 Additional Results on DDQN

In addition to the final performance shown in the main text, we show in Table 1 numerical results from all experiments. Figure 5 shows learning curves from all experiments.

Table 1: Performance in all three environments, taken at 800k timesteps (Pursuit), 300k timesteps (Battle), 300k timesteps (Adv. Pursuit). Numbers in parentheses indicate standard deviation. Highest performance in each environment is printed in bold.

	pursuit	battle	adversarial_pursuit
ddqn	181.43 (+-4.16)	9.46 (+-0.86)	-719.78 (+-66.82)
ddqn_paramshare	-139.03 (+-121.11)	15.15 (+-8.64)	**-268.78 (+-60.11)**
ddqn_shareall	148.27 (+-9.22)	9.25 (+-6.60)	-568.91 (+-40.82)
maddpg	-342.24 (+-4.12)	-10.31 (+-nan)	-1071.71 (+-8.56)
seac	32.51 (+-70.27)	-2.73 (+-0.03)	-993.40 (+-120.22)
super_ddqn_gaussian	373.63 (+-9.15)	16.28 (+-3.42)	-480.37 (+-15.79)
super_ddqn_quantile	**454.56 (+-5.89)**	**17.05 (+-2.74)**	-506.29 (+-35.03)
super_ddqn_stochastic	266.42 (+-85.52)	7.43 (+-5.64)	-582.59 (+-15.20)

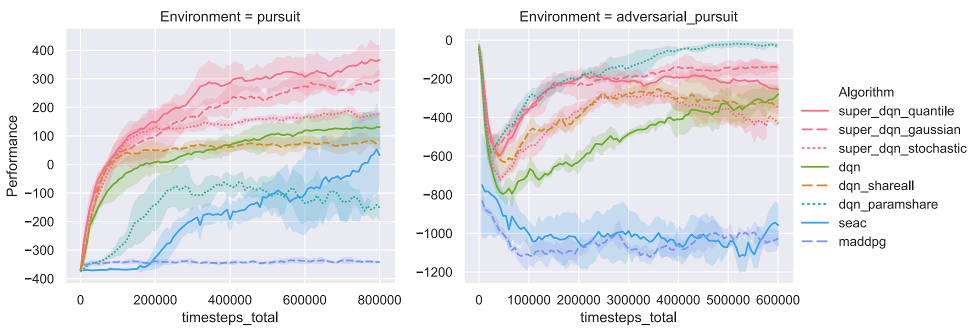

Figure 6: Performance of SUPER-DQN variants with target bandwidth 0.1 on Pursuit and Adversarial-Pursuit. For Pursuit, performance is the total mean episode reward from all agents. For Adversarial-Pursuit, performance is the total mean episode reward from all agents in the prey team. Shaded areas indicate one standard deviation.

Figure 7: Performance of SUPER-dueling-DDQN variants with target bandwidth 0.1 on all Battle and Adversarial=Pursuit, with co-evolving opponents. Performance is the total mean episode reward from all agents in the sharing team (blue team in Battle, prey team in Adversarial-Pursuit). Shaded areas indicate one standard deviation.

C.2 DQN and Dueling DDQN

For all of the DDQN and SUPER-DDQN variants discussed in Section 5, we consider also variants based on standard DQN. Figure 6 shows results from these experiments.

C.3 Co-Evolving Teams

In Battle and Adversarial-Pursuit we further show a variant where the opposing team are co-evolving with the blue / prey team. In this variant, all agents start from a randomly initialized policy and train concurrently, using a DDQN algorithm. However, only the blue / prey team share experiences using the SUPER mechanism. We only do this for the DDQN baseline as well as discriminate and share-all SUPER variants. This is in part because some of the other baseline algorithms do not support concurrently training opposing agents with a different algorithm in available implementations; and in part because we consider this variant more relevant to real-world scenarios where fully centralized training may not be feasible. We aim to show here how sharing even a small number of experiences changes the learning dynamics versus to non-sharing opponents. Figure 7 shows this variant.

C.4 Further Ablations

In addition to the ablations presented in the main text, we include here additional results from the Battle environment in Figure 9. Results are broadly similar to the results in the main text, with a notably bad performance for uniform random experience sharing.

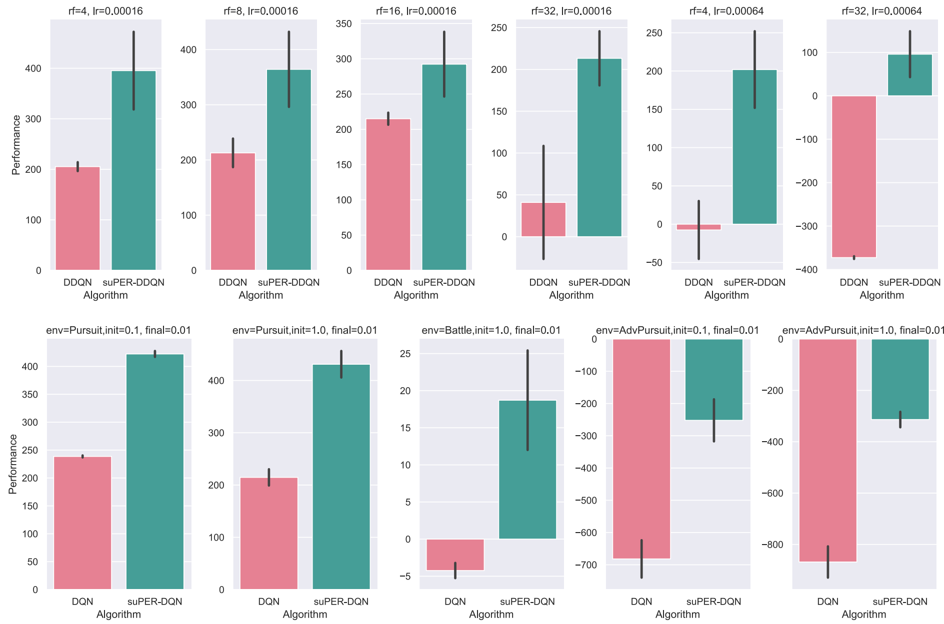

Figure 8: Performance of DDQN and SUPER-DDQN (gaussian experience selection, target bandwidth 0.1) for differing hyperparameter settings of the underlying DDQN algorithm. Top: Different learning rates and rollout fragment lengths in Pursuit. Bottom: Different exploration settings in Pursuit and co-evolving variants of Batle and Adversarial-Pursuit. Hyperparameters otherwise identical to those used in Figure 1. Performance measured at 1M timesteps in Pursuit, 300k timesteps in Battle, 400k timesteps in Adversarial-Pursuit.

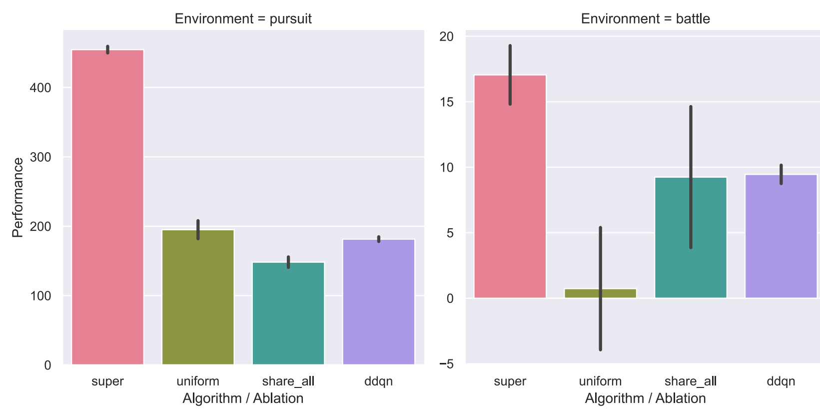

Figure 9: Performance of quantile SUPER vs share-all and uniform random experience sharing in Pursuit at 800k timesteps.

D Stability Across Hyperparameters

Figure 8 shows performance of no-sharing DDQN and SUPER-DDQN for different hyperparameters. As we can see, SUPER-DDQN outperforms no-sharing DDQN consistently across all the hyperparameter settings considered.

E Additional Analysis of Bandwidth Sensitivity

We present here a more detailed analysis of bandwidth sensitivity of SUPER-DDQN in the three experience selection modes we discuss in the main text. Figure 10 shows the mean performance

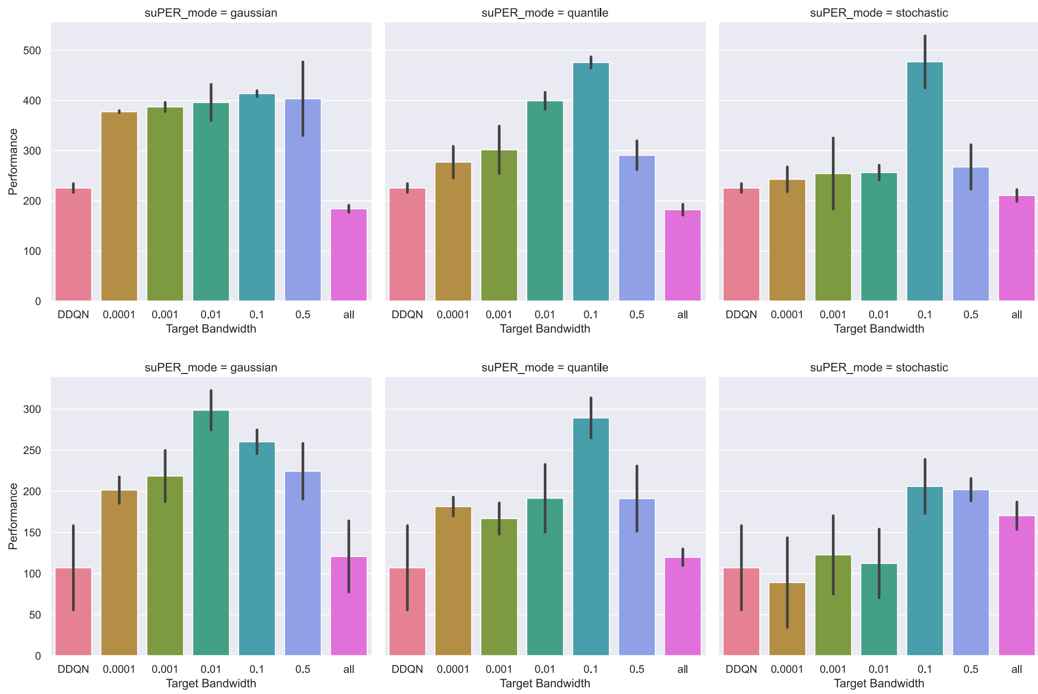

Figure 10: Performance of SUPER with different experience selection and varying bandwidth in Pursuit at 1-2M timesteps (top) and at 250k timesteps (bottom).

across five seeds for gaussian (left), quantile (middle) and stochastic (right) experience selection, at 1-2M timesteps (top) and at 250k timesteps (bottom). We can see that at 1-2M timesteps and a target bandwidth of 0.1, all three experience selection criteria perform similary. One thing that stands out is that stochastic selection has much lower performance at other target bandwidths, and also much less performance uplift compared to no-sharing DDQN at 250k timesteps at any bandwidth. Gaussian experience selection appears to be less sensitive to target bandwidth, but upon closer analysis we found that it also was much less responsive in terms of how much actual bandwidth it used at different settings. Figure 11 (left) shows the actual bandwidth used by each selection criterion at different target bandwidths. We can see that quantile and stochastic experience hit their target bandwidth very well in general.[5] What stands out, however, is that gaussian selection vastly overshoots the target bandwidth at lower settings, never going significantly below 0.01 actual bandwidth.

What is a fairer comparison therefore is to look at performance versus actual bandwidth used for each of the approaches, which we do in Figure 11 (middle, at 1-2M timesteps, and right, at 250k timesteps). For these figures, we did the following: First, for each experience selection approach and target bandwidth, we computed the mean performance and mean actual bandwidth across the five seeds. Then, for each experience selection mode, we plotted these $(\mathrm{meanactualbandwidth}, \mathrm{meanperformance})$ (one for each target bandwidth) in a line plot.[6] The result gives us a rough estimate of how each approach's performance varies with actual bandwidth used. We see again that stochastic selection shows worse performance than quantile at low bandwidths, and early in training. We also see that gaussian selection very closely approximates quantile selection. Notice that gaussian selection never hits an exact actual bandwidth of 0.1, and so we cannot tell from these data if it would match quantile selection's performance at its peak. However, we can see that at the actual bandwidths that gaussian selection does hit, it shows very similar performance to quantile selection. As stated in the main

[5] Quantile selection overshoots at 1e-4 (0.0001) target bandwidth and is closer to 1e-3 actual bandwidth usage, which we attribute to a rounding error, as we ran these experiments with a window size of 1500 (1.5e+3), and a quantile of less than a single element is not well-defined.

[6] Because each data point now has variance in both x- and y-directions, it is not possible to draw error bars for these.

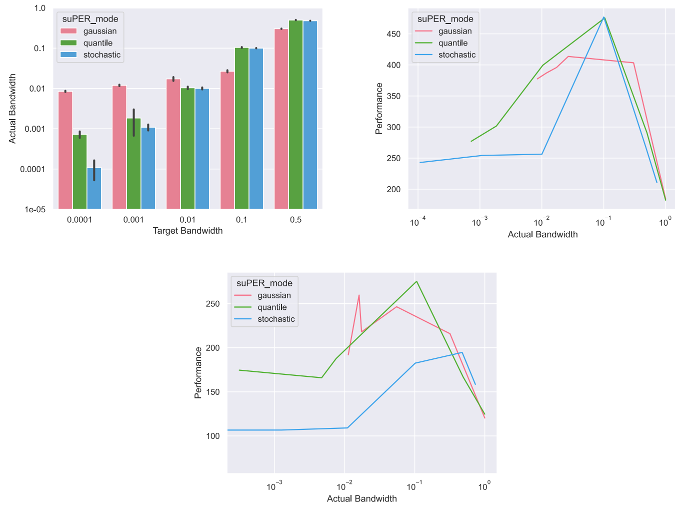

Figure 11: Left: Actual bandwidth used (fraction of experiences shared) at different target bandwidths. Middle, right: Performance compared to actual bandwidth used at 1-2M and 250k timesteps.

text, our interpretation of this is that using mean and variance to approximate the exact distribution of absolute td-errors is a reasonable approximation, but that we might need to be more clever in selecting c in equation 3.

F Experiment Hyperparameters & Details

We performed all experiments using the open-source library **RLlib** [16]. Experiments in Figure 1 and 6 were ran using RLlib version 2.0.0; experiments in other figures were run using version 1.13.0. Environments used are from PettingZoo [33], including SISL [8] and MAgent [40]. The SUPER algorithm was implemented by modifying RLlib's standard DQN algorithm to perform the SUPER experience sharing between rollout and training. Table 2 lists all the algorithm hyperparameters and environment settings we used for all the experiments. Experiments in the "Stability across hyperparameters" section had hyperparameters set to those listed in Table 2 except those specified in Figure 8. Any parameters not listed were left at their default values. Hyperparameters were tuned using a grid search; some of the combinations tested are also discussed in the "Stability across hyperparameters" section. For DQN, DDQN and their SUPER variants, we found hyperparameters using a grid search on independent DDQN in each environment, and then used those hyperparameters for all DQN/DDQN and SUPER variants in that environment. For all other algorithms we performed a grid search for each algorithm in each environment. For MADDPG we attempted further optimization using the Python **HyperOpt** package [4], however yielding no significant improvement over our manual grid search. For SEAC, we performed a grid search in each environment, but found no better hyperparameters than the default. We found a CNN network architecture using manual experimentation in each environment, and then used this architecture for all algorithms except MADDPG where we used a fully connected net for technical reasons. We tested all other algorithms using both the hand-tuned CNN as well as a fully connected network, and found that the latter performed significantly worse, but still reasonable (and in particular significantly better than MADDPG using the same fully connected network, on all domains).

All experiments were repeated with three seeds. All plots show the mean and standard deviation of these seeds at each point in training. For technical reasons, individual experiment runs did not always report data at identical intervals. For instance, one run might report data when it had sampled 51000 environment timesteps, and another run might report at 53000 environment timesteps. In order to still be able to report a meaningful mean and standard deviation across repeated runs, we rounded down the timesteps reported to the nearest k steps, i.e. taking both the data above to represent each run's performance at 50000 steps. We set k to the target reporting interval in each domain (8000 timesteps in Pursuit, 6000 timesteps in the other two domains). Where a run reported more than once in a

10000 step interval, we took the mean of its reports to represent that run's performance in the interval. Mean and standard deviation were calculated across this mean performance for each of the five seeds. To increase legibility, we applied smoothing to Figures 5 and 6 using an exponential window with $\alpha = 0.3$ for Pursuit, $\alpha = 0.1$ for Battle, and $\alpha = 0.25$ for Adversarial-Pursuit. This removes some noise from the reported performance, but does not change the relative ordering of any two curves.

G Implementation & Reproducibility

All source code is included in the supplementary material and will be made available on publication under an open-source license. We refer the reader to the included README file, which contains instructions to recreate the experiments discussed in this paper.

Table 2: Hyperparameter Configuration Table - SISL: Pursuit

Environment Parameters			
HyperParameters	**Value**	**HyperParameters**	**Value**
max cycles	500	x/y sizes	16/16
shared reward	False	num evaders	30
horizon	500	n catch	2
surrounded	True	num agents(pursuers)	8
tag reward	0.01	urgency reward	-0.1
constrained window	1.0	catch rewards	5
obs range	7		
CNN Network			
CNN layers	[32,64,64]	Kernel size	[2,2]
Strides	1		
SUPER / DQN / DDQN			
learning rate	0.00016	final exploration epsilon	0.001
batch size	32	nframework	torch
prioritized replay_alpha	0.6	prioritized replay eps	1e-06
dueling	True	target network update_freq	1000
buffer size	120000	rollout fragment length	4
initial exploration epsilon	0.1		
MADDPG			
Actor lr	0.00025	Critic lr	0.00025
NN(FC)	[64,64]	tau	0.015
framework	tensorflow	actor feature reg	0.001
SEAC			
learning rate	3e-4	adam eps	0.001
batch size	5	use gae	False
framework	torch	gae lambda	0.95
entropy coef	0.01	value loss coef	0.5
max grad norm	0.5	use proper time limits	True
recurrent policy	False	use linear lr decay	False
seac coef	1.0	num processes	4
num steps	5		

Table 3: Hyperparameter Configuration Table- MAgent: Battle

HyperParameters	Value	HyperParameters	Value
Environment Parameters			
minimap mode	False	step reward	-0.005
Num blue agents	6	Num red agents	6
dead penalty	-0.1	attack penalty	-0.1
attack opponent reward	0.2	max cycles	1000
extra features	False	map size	18
CNN Network			
CNN layers	[32,64,64]	Kernel size	[2,2]
Strides	1		
SUPER / DQN / DDQN			
learning rate	1e-4	batch size	32
framework	torch	prioritized replay_alpha	0.6
prioritized replay eps	1e-06	horizon	1000
dueling	True	target network update_freq	1200
rollout fragment length	5	buffer size	90000
initial exploration epsilon	0.1	final exploration epsilon	0.001
MADDPG			
Actor lr	0.00025	Critic lr	0.00025
NN(FC)	[64,64]	tau	0.015
framework	tensorflow	actor feature reg	0.001
SEAC			
learning rate	3e-4	adam eps	0.001
batch size	5	use gae	False
framework	torch	gae lambda	0.95
entropy coef	0.01	value loss coef	0.5
max grad norm	0.5	use proper time limits	True
recurrent policy	False	use linear lr decay	False
seac coef	1.0	num processes	4
num steps	5		

Table 4: Hyperparameter Configuration Table - MAgent: Adversarial Pursuit

Environment Parameters			
HyperParameters	**Value**	**HyperParameters**	**Value**
Number predators	4	Number preys	8
minimap mode	False	tag penalty	-0.2
max cycles	500	extra features	False
map size	18		
Policy Network			
CNN layers	[32,64,64]	Kernel size	[2,2]
Strides	1		
SUPER / DQN / DDQN			
learning rate	1e-4	batch size	32
framework	torch	prioritized replay alpha	0.6
prioritized replay eps	1e-06	horizon	500
dueling	True	target network update_freq	1200
buffer size	90000	rollout fragment length	5
initial exploration epsilon	0.1	final exploration epsilon	0.001
MADDPG			
Actor lr	0.00025	Critic lr	0.00025
NN(FC)	[64,64]	tau	0.015
framework	tensorflow	actor feature reg	0.001
SEAC			
learning rate	3e-4	adam eps	0.001
batch size	5	use gae	False
framework	torch	gae lambda	0.95
entropy coef	0.01	value loss coef	0.5
max grad norm	0.5	use proper time limits	True
recurrent policy	False	use linear lr decay	False
seac coef	1.0	num processes	4
num steps	5		