A. Formal derivation of the illustrative example in Section 2

A VDN decomposition of the joint state-action-value function in the simplified predator-prey task shown in Figure 1 is:

$$Q^\pi(s,u^1,u^2) \quad = \quad \mathbb{E}\Big[r(s,u^1,u^2)+\gamma V^\pi(s') \,\Big|\, \substack{u^2\sim\pi^2(\cdot|s) \\ s'\sim P(\cdot|s,u^1,u^2)}\Big] \quad \approx \quad Q^1(s,u^1)+Q^2(s,u^2) \quad =: \quad Q_{\text{tot}}(s,u^1,u^2)\,.$$

Here $V^\pi(s')$ denotes the expected return of state $s'\in\mathcal{S}$. However, in this analysis we are mainly interested in the behavior at the beginning of training, and so we assume in the following that the estimated future value is close to zero, i.e. $V^\pi(s')\approx 0, \forall s'\in\mathcal{S}$. The VDN decomposition allows to derive a decentralized policy $\pi(u^1,u^2|s) := \pi^1(u^1|s)\,\pi^2(u^2|s)$, where π^i is usually an ϵ-greedy policy based on agent i's utility $Q^i(s,u^i)$, $i\in\{1,2\}$. Similar to Rashid et al. (2020a), we analyze here the corresponding VDN projection operator:

$$\Pi^\pi_{\text{vdn}}[Q^\pi] \quad := \quad \operatorname*{arg\,min}_{Q_{\text{tot}}} \sum_{u^1\in\mathcal{U}^1}\sum_{u^2\in\mathcal{U}^2} \pi(u^1,u^2|s)\Big(Q_{\text{tot}}(s,u^1,u^2)-Q^\pi(s,u^1,u^2)\Big)^2\,.$$

Setting the gradient of the above mean-squared loss to zero yields for Q^1 (and similarly for Q^2):

$$Q^1(s,u^1) \quad \stackrel{!}{=} \quad \sum_{u^2\in\mathcal{U}^2}\pi^2(u^2|s)\Big(\underbrace{r(s,u^1,u^2)+\gamma\mathbb{E}[\,V^\pi(s')\,|\,{}_{s'\sim P(\cdot|s,u^1,u^2)}]}_{Q^\pi(s,u^1,u^2)} - Q^2(s,u^2)\Big)\,.$$

In the following we use $\bar{V}_{u^1} := \max_{u^2}\mathbb{E}[\,V^\pi(s')\,|\,{}_{s'\sim P(\cdot|s,u^1,u^2)}]$. Relative overgeneralization (Panait et al., 2006) occurs when an agent's utility (for example agent 1's) of the catch action $u^1=C$ falls below the movement actions $u^1\in\{L,R\}$:

$$\begin{aligned}
& Q^1(s,C) && < && Q^1(s,L/R) \\
\Leftrightarrow \quad & \sum_{u^2\in\mathcal{U}^2}\pi^2(u^2|s)\,\big(r(s,C,u^2)-Q^2(s,u^2)\big) && < && \sum_{u^2\in\mathcal{U}^2}\pi^2(u^2|s)\,\big(Q^\pi(s,L/R,u^2)-Q^2(s,u^2)\big) \\
\Rightarrow \quad & r\,\pi^2(C|s)-p\,\pi^2(L|s)-p\,\pi^2(R|s) && < && -p\,\pi^2(C|s)\;+\;\gamma\big(1-\pi^2(C|s)\big)\bar{V}_{L/R} \\
\Leftrightarrow \quad & r\,\pi^2(C|s) && < && p\,\big(\pi^2(L|s)+\pi^2(R|s)-\pi^2(C|s)\big)\;+\;\gamma\big(1-\pi^2(C|s)\big)\bar{V}_{L/R} \\
\Leftrightarrow \quad & r\,\pi^2(C|s) && < && p\,\big(1-2\pi^2(C|s)\big)\;+\;\gamma\big(1-\pi^2(C|s)\big)\bar{V}_{L/R} \\
\Leftrightarrow \quad & r && < && p\,\big(\tfrac{1}{\pi^2(C|s)}-2\big)\;+\;\gamma\big(1-\pi^2(C|s)\big)\bar{V}_{L/R}
\end{aligned}$$

This demonstrates that, at the beginning of training with $\bar{V}_{L/R}=0$, relative overgeneralization occurs at $p>r$ for $\pi^2(C|s)=\frac{1}{3}$, at $p>\frac{3}{2}\,r$ for $\pi^2(C|s)=\frac{3}{8}$, at $p>3\,r$ for $\pi^2(C|s)=\frac{3}{7}$, and at *never* at $\pi^2(C|s)>\frac{1}{2}$, as shown in Figure 2. Note that the value $\bar{V}_{L/R}$ contributes a constant w.r.t. punishment p, that is scaled by $1-\pi^2(C|s)$, and that positive values make the problem harder (require lower p). An initial relative overgeneralization will make agents choose the wrong greedy actions, which lowers $\pi^2(C|s)$ even further and therefore solidifies the pathology when ϵ get's annealed. Empirical thresholds in experiments can differ significantly, though, as random sampling increases the chance to over/underestimate $\pi^2(C|s)$ during the exploration phase and the estimated value $V^\pi(s')$ has a non-trivial influence. Furthermore, the above thresholds cannot be transferred directly to the experiments conducted in Section 6, which are much more challenging: (i) it requires 3 agents to catch a prey, (ii) agents have to explore 5 actions, (iii) agents can execute catch actions when they are alone with the prey, and (iv) catch actions do not end the episode, increasing the influence of empirically estimated $V^\pi(s')$. Empirically we observe that VDN fails to solve tasks somewhere between $p=0.004\,r$ and $p=0.008\,r$. The presented analysis provides therefore only an illustrative example how UneVEn can help to overcome relative overgeneralization.

B. Training algorithm

Algorithm 1 and 2 presents the training of MAUSFs with UneVEn. Our method is able to learn on all tasks (target $\boldsymbol{w}$ and sampled $\boldsymbol{z}$) simultaneously in a sample efficient manner using the same feature $\boldsymbol{\phi}_t \equiv \boldsymbol{\phi}(s_t,\boldsymbol{u}_t)$ due to the linearly decomposed reward function (Equation 1).

C. Experimental Domain Details and Analysis

C.1. Predator-Prey

We consider a complicated partially observable predator-prey (PP) task in an 10×10 grid involving eight agents (predators) and three prey that is designed to test coordination between agents, as each prey needs a simultaneous *capture* action by

Algorithm 1 Training MAUSFs with UneVEn

Require ϵ, α, β target task $\boldsymbol{w}$, set of agents $\mathcal{A}$, standard deviation σ
Initialize the local-agent SF network $\boldsymbol{\psi}^a(\tau^a, u^a, \boldsymbol{z}; \theta)$ and replay buffer $\mathcal{M}$
for fixed number of **epochs do**
  $\nu \sim \mathcal{N}(\boldsymbol{w}, \sigma \mathbf{I}_d)$; $\boldsymbol{o}_0 \leftarrow$ RESETENV()
  $t = 0$; $\mathcal{M} \leftarrow$ NEWEPISODE$(\mathcal{M}, \nu, \boldsymbol{o}_0)$
  while not **terminated do**
    if Bernoulli(ϵ)=1 then $\boldsymbol{u}_t \leftarrow$ Uniform($\boldsymbol{\mathcal{U}}$)
    else $\boldsymbol{u}_t \leftarrow$ UNEVEN$(\boldsymbol{\tau}_t, \nu)$
    $\langle \boldsymbol{o}_{t+1}, \boldsymbol{\phi}_t \rangle \leftarrow$ ENVSTEP$(\boldsymbol{u}_t)$
    $\mathcal{M} \leftarrow$ ADDTRANSITION$\big(\mathcal{M}, \boldsymbol{u}_t, \boldsymbol{o}_{t+1}, \boldsymbol{\phi}_t\big)$
    $t \leftarrow t + 1$
  end while
  $\mathcal{L} \leftarrow 0$; $\mathcal{B} \leftarrow$ SAMPLEMINIBATCH$(\mathcal{M})$
  for all $\{\boldsymbol{\tau}_t, \boldsymbol{u}_t, \boldsymbol{\phi}_t, \boldsymbol{\tau}_{t+1}, \nu\} \in \mathcal{B}$ **do**
    for all $\boldsymbol{z} \in \nu \cup \{\boldsymbol{w}\}$ **do**
      $\boldsymbol{u}'_{\boldsymbol{z}} \leftarrow \big\{ \arg\max_{u \in \mathcal{U}} \boldsymbol{\psi}^a(\tau^a_{t+1}, u, \boldsymbol{z}; \theta)^\top \boldsymbol{z} \big\}_{a \in \mathcal{A}}$
      $\mathcal{L} \leftarrow \mathcal{L} + \big\| \boldsymbol{\phi}_t + \gamma \boldsymbol{\psi}_{tot}(\boldsymbol{\tau}_{t+1}, \boldsymbol{u}'_{\boldsymbol{z}}, \boldsymbol{z}; \theta^-) - \boldsymbol{\psi}_{tot}(\boldsymbol{\tau}_t, \boldsymbol{u}_t, \boldsymbol{z}; \theta) \big\|_2^2$
    end for
  end for
  $\theta \leftarrow$ OPTIMIZE$(\theta, \nabla_\theta \mathcal{L})$
  $\theta^- \leftarrow (1 - \beta)\, \theta^- + \beta\, \theta$
end for

Algorithm 2 UNEVEN$(\boldsymbol{\tau}_t, \nu)$

if Bernoulli$(\alpha) = 1$ **or** Scheme is **Target then**
  $\mathcal{C}_2 \leftarrow \{\boldsymbol{w}\}$
else
  if Scheme is **Uniform then**
    $\mathcal{C}_2 \leftarrow \nu \sim$ Uniform(ν)
  else if Scheme is **Greedy then**
    $\mathcal{C}_2 \leftarrow \nu \cup \{\boldsymbol{w}\}$
  end if
end if
if Use_GPI_Policy is **True then**
  $\mathcal{C}_1 \leftarrow \nu \cup \{\boldsymbol{w}\}$
  $\boldsymbol{u}_t \leftarrow \{u^a_t = \arg\max_{u \in \mathcal{U}} \max_{\boldsymbol{k} \in \mathcal{C}_2} \max_{\boldsymbol{z} \in \mathcal{C}_1} \boldsymbol{\psi}^a(\tau^a_t, u, \boldsymbol{z}; \theta)^\top \boldsymbol{k}\}_{a \in \mathcal{A}}$
else
  $\boldsymbol{u}_t \leftarrow \{u^a_t = \arg\max_{u \in \mathcal{U}} \max_{\boldsymbol{k} \in \mathcal{C}_2} \boldsymbol{\psi}^a(\tau^a_t, u, \boldsymbol{k}; \theta)^\top \boldsymbol{k}\}_{a \in \mathcal{A}}$
end if
return $\boldsymbol{u}_t$

A1-Capture		
	A2-Capture	A2-Other
A3-Capture	+1	$-p$
A3-Other	$-p$	$-p$

A1-Other		
	A2-Capture	A2-Other
A3-Capture	$-p$	$-p$
A3-Other	$-p$	0

Table 1. Joint-Reward function of three agents surrounding a prey. The first table indicates joint-rewards when Agent 1 takes capture action and second table indicates joint-rewards when Agent 1 takes any other action. Notice that there are numerous joint actions leading to penalty p.

at least three surrounding agents to be captured. Each agent can take 6 actions i.e. move in one of the 4 directions (Up, Left, Down, Right), remain still (no-op), or try to catch (capture) any adjacent prey. The prey moves around in the grid with a probability of 0.7 and remains still at its position with probability 0.3. Impossible actions for both agents and prey are marked unavailable, for eg. moving into an occupied cell or trying to take a capture action with no adjacent prey.

If either a single or a pair of agents take a capture action on an adjacent prey, a negative reward of magnitude p is given. If three or more agents take the capture action on an adjacent prey, it leads to a successful capture of that prey and yield a positive reward of $+1$. The maximum possible reward for capturing all prey is therefore $+3$. Each agent observes a 5×5 grid centered around its position which contains information showing other agents and prey relative to its position. An episode ends if all prey have been captured or after 800 time steps. This task is similar to one proposed by Böhmer et al. (2020); Son et al. (2019), but significantly more complex in terms of the coordination required amongst agents as more agents need to coordinate simultaneously to capture the prey.

This task is challenging for two reasons. First, depending on the magnitude of p, exploration is difficult as even if a single agent miscoordinates, the penalty is given, and therefore, any steps toward successful coordination are penalized. Second, the agents must be able to differentiate between the values of successful and unsuccessful collaborative actions, which monotonic value functions fail to do on tasks exhibiting RO.

Proposition. For, the predator-prey game defined above, the optimal joint action reward function for any group of $2 \leq k \leq n$ predator agents surrounding a prey is *nonmonotonic* (as defined by Mahajan et al., 2019) iff $p > 0$.

Proof. Without loss of generality, we assume a single prey surrounded by three agents (A_1, A_2, A_3) in the environment. The joint reward function for this group of three agents is defined in Table 1.

For the case $p > 0$ the proposition can be easily verified using the definition of non-monotonicity (Mahajan et al., 2019). For any $3 \leq k \leq n$ agents attempting to catch a prey in state s, we fix the actions of any $k-3$ agents to be "*other*" indicating either of up, down, left, right, and noop actions and represent it with $\boldsymbol{u}^{k-3}$. Next we consider the rewards r for two cases:

- If we fix the action of any *two* of the remaining three agents as "other" represented as $\boldsymbol{u}^2$, the action of the remaining agent becomes $u^1 = \arg\max_{u \in \mathcal{U}} r(s, \langle u, \boldsymbol{u}^2, \boldsymbol{u}^{k-3} \rangle) =$ "other".
- If we fix the $\boldsymbol{u}^2$ to be "capture", we have : $u_1 = \arg\max_{u \in \mathcal{U}} r(s, \langle u, \boldsymbol{u}^2, \boldsymbol{u}^{k-3} \rangle) =$ "capture".

Thus the best action for agent A_1 in state s depends on the actions taken by the other agents and the rewards $R(s)$ are non-monotonic. Finally for the equivalence, we note that for the case $p = 0$ we have that a default action of "capture" is always optimal for any group of k predators surrounding the prey. Thus the rewards are monotonic as the best action for any agent is independent of the rest. □

C.2. m-Step Matrix Games

Figure 9 shows the m-step matrix game for $m = 10$ from Mahajan et al. (2019), where there are $m-2$ intermediate steps, and selecting a joint-action with zero reward leads to termination of the episode.

C.3. StarCraft II Micromanagement

We use the negative reward version of the SMAC benchmark (Samvelyan et al., 2019) where each ally agent unit is additionally penalized (penalty p) for being killed or suffering damage from the enemy, in addition to receiving positive reward for killing/damage on enemy units, which has recently been shown to improve performance (Son et al., 2020). We

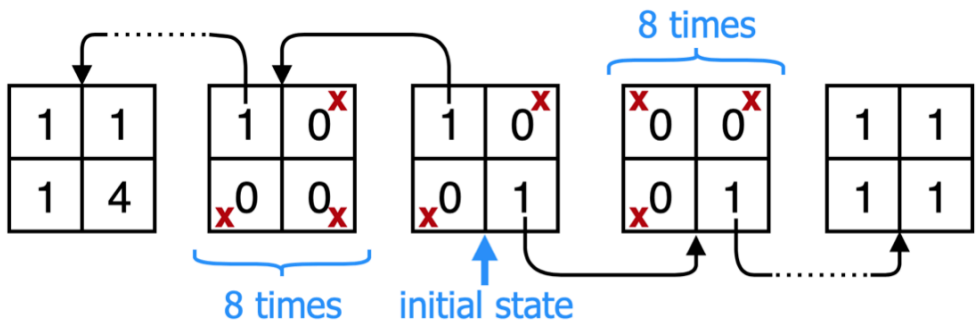

Figure 9. m-step matrix game from Mahajan et al. (2019) for $m = 10$. The red cross means that selecting that joint action will lead to termination of the episode.

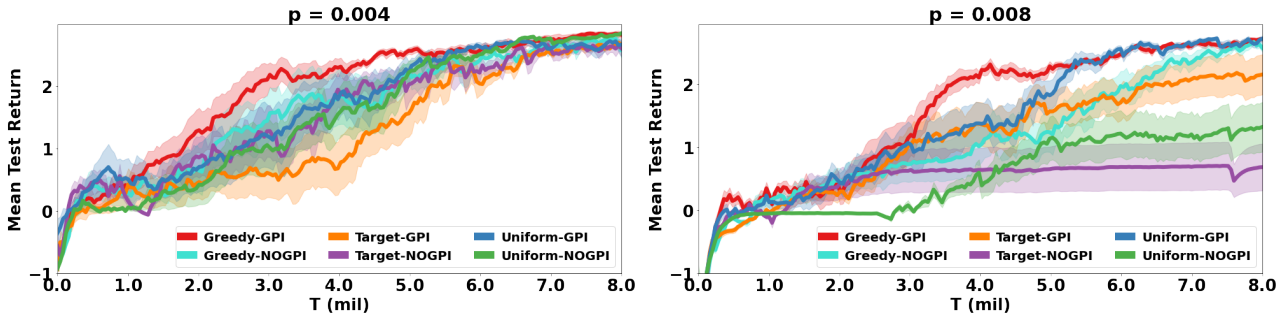

Figure 10. Additional Ablation results: Comparison between different action selection of UneVEn for $p \in \{0.004, 0.008\}$.

consider two versions of the penalty p, i.e. $p = 0.5$ which is default in the SMAC benchmark and $p = 1.0$ which equally weights the lives of allies and enemies, making the task more prone to exhibiting RO.

D. Implementation Details

D.1. Hyper parameters

All algorithms are implemented in the PyMARL framework (Samvelyan et al., 2019). All our experiments use ϵ-greedy scheme where ϵ is decayed from $\epsilon = 1$ to $\epsilon = 0.05$ over $\{250k, 500k\}$ time steps. All our tasks use a discount factor of $\gamma = 0.99$. We freeze the trained policy every $30k$ timesteps and run 20 evaluation episodes with $\epsilon = 0$. We use learning rate of 0.0005 with soft target updates for all experiments. We use a target network similar to Mnih et al. (2015) with "soft" target updates, rather than directly copying the weights: $\theta^- \leftarrow \beta * \theta + (1 - \beta) * \theta^-$, where θ are the current network parameters. We use $\beta = 0.005$ for PP and m-step experiments and $\beta = 0.05$ for SC2 experiments. This means that the target values are constrained to change slowly, greatly improving the stability of learning. All algorithms were trained with RMSprop optimizer by one gradient step on loss computed on a batch of 32 episodes sampled from a replay buffer containing last 1000 episodes (for SC2, we use last 3000 episodes). We also used gradient clipping to restrict the norm of the gradient to be ≤ 10.

The probability α of action selection based on target task in UneVEn with uniform and greedy action selection schemes increases from $\alpha = 0.3$ to $\alpha = 1.0$ over $\{250k, 500k\}$ time steps. For sampling related tasks using normal distribution, we use $\mathcal{N}(\boldsymbol{w}, \sigma \mathbf{I}_d)$ centered around target task $\boldsymbol{w}$ with $\sigma \in \{0.1, 0.2\}$. At the beginning of each episode, we sample *six* related tasks, therefore $|\nu| = 6$ (for SC2, we use $|\nu| = 3$).

D.2. NN Architecture

Each agent's local observation o_t^a are concatenated with agent's last action u_{t-1}^a, and then passed through a fully-connected (FC) layers of 128 neurons (for SC2, we use 1024 neurons), followed by ReLU activation, a GRU (Chung et al., 2014), and another FC of the same dimensionality to generate a action-observation history summary for the agent. Each agent's

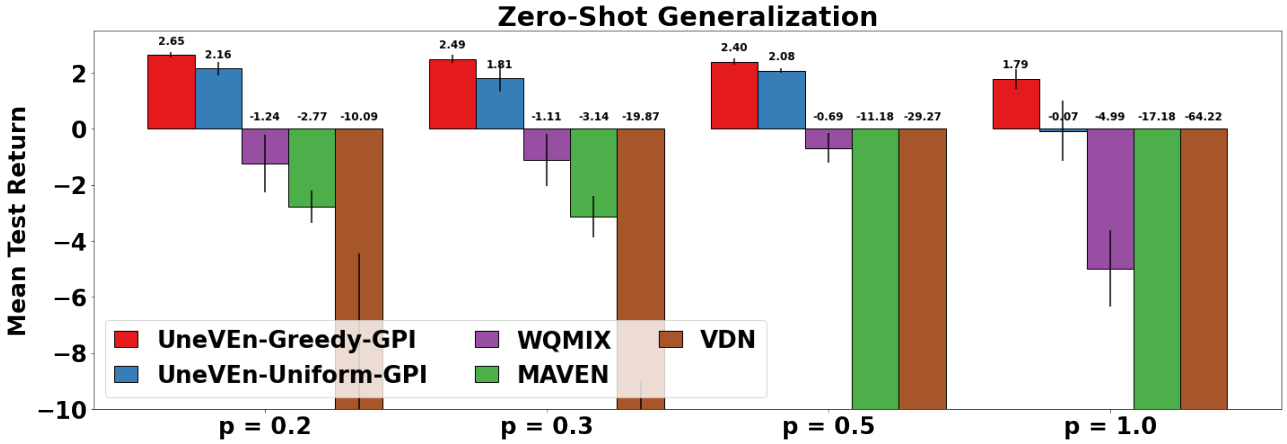

Figure 11. Additional Zero-shot generalization results for $p \in \{0.2, 0.3, 0.5, 1.0\}$.

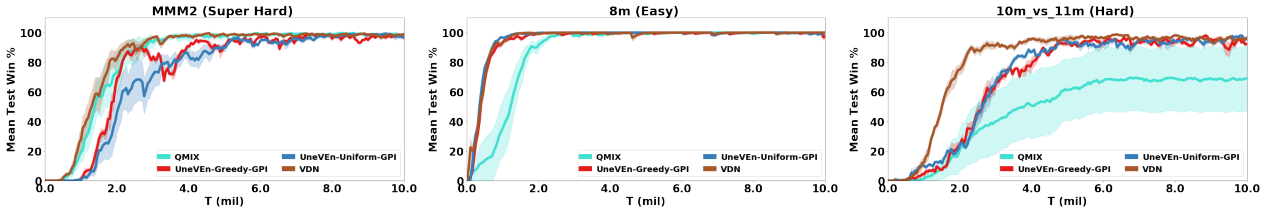

Figure 12. Comparison between UneVEn, VDN and QMIX on SMAC maps with penalty $p = 0.5$.

task vector $\boldsymbol{z} \in \nu \cup \{\boldsymbol{w}\}$ is passed through a FC layer of 128 neurons (for SC2, we use 1024 neurons) followed by ReLU activation to generate an internal task embedding. The history and task embedding are concatenated together and passed through two hidden FC-256 layers (for SC2, FC-2048 layer) and ReLU activations to generate the outputs for each action. For methods with non-linear mixing such as QMIX (Rashid et al., 2020b), WQMIX (Rashid et al., 2020a), and MAVEN (Mahajan et al., 2019), we adopt the same hypernetworks from the original paper and test with either a single or double hypernet layers of dim 64 utilizing an ELU non-linearity. For all baseline methods, we use the code shared publicly by the corresponding authors on Github.

E. Additional Results

E.1. Predator-Prey

Figure 10 presents additional ablation results for comparison between UneVEn with different action selection schemes for $p \in \{0.004, 0.008\}$. Figure 11 presents additional zero-shot generalization results for policies trained on target task with penalty $p = 0.004$ tested on tasks with penalty $p \in \{0.2, 0.3, 0.5, 1.0\}$. For UneVEn-Greedy-GPI, we can observe that the average number of miscoordinated capture attempts per episode actually drops with p and converges around 1.2, i.e., for return R_p, average mistakes per episode is $\frac{3-R_p}{p} = \{1.75, 1.7, 1.2, 1.2\}$ for $p \in \{0.2, 0.3, 0.5, 1.0\}$.

E.2. StarCraft II Micromanagement

We first discuss different reward functions in the SMAC benchmark (Samvelyan et al., 2019). The default SMAC reward function depends on three major components: (1) `delta_enemy`: accumulates difference in health and shield of all enemy units between last time step and current time step, (2) `delta_ally`: accumulates difference in health and shield of all ally units between last time step and current time step scaled by `reward_negative_scale`, (3) `delta_deaths`: defined below.

For the original reward function `reward_only_positive = True`, `delta_deaths` is defined as positive reward

of reward_death_value for every enemy unit killed. The final reward is abs(delta_enemy + delta_deaths). Notice that some of the units have shield regeneration capabilities and therefore, delta_enemy might contain negative values as current time step health of enemy unit might be higher than last time step. Therefore, to enforce positive rewards, abs function is used.

For reward_only_positive = False, delta_deaths is defined as positive reward of reward_death_value for every enemy unit killed, and penalizes reward_negative_scale * reward_death_value for every ally unit killed. The final reward is simply: delta_enemy + delta_deaths - delta_ally.

Notice that reward_negative_scale measures the relative importance of lives of ally units compared to enemy units. For reward_negative_scale = 1.0, both enemy and ally units lives are equally valued, for reward_negative_scale = 0.5, ally units are only valued half of enemy units, and for reward_negative_scale = 0.0, ally units lives are not valued at all. However, reward_only_positive = False with reward_negative_scale = 0.0 is NOT the same as setting reward_only_positive = True as the latter uses an abs function.

To summarize, reward_only_positive decides whether there is an additional penalty for health reduction and death of ally units, and reward_negative_scale determines the relative importance of lives of ally units when reward_only_positive = False. Figure 13 shows that for most of the maps, there is not a big performance difference for VDN and QMIX between different reward functions (original, $p = 0.0$ and $p = 0.5$). However, for some maps using reward_only_positive = False with either $p = 0.0$ and $p = 0.5$ improves performance over the original reward function. We hypothesize that the use of abs in the original reward function can detriment the learning of agent as it might get positive absolute reward to increase the health of enemy units.

Figure 12 presents the mean test win rate for SMAC maps with reward_only_positive = False and low penalty of $p = 0.5$. Both VDN and QMIX achieve almost 100% win rate on these maps and the additional complexity of learning MAUSFs in our approach results in slightly slower convergence. However, UneVEn with both GPI schemes matches the performance as VDN and QMIX in all maps.

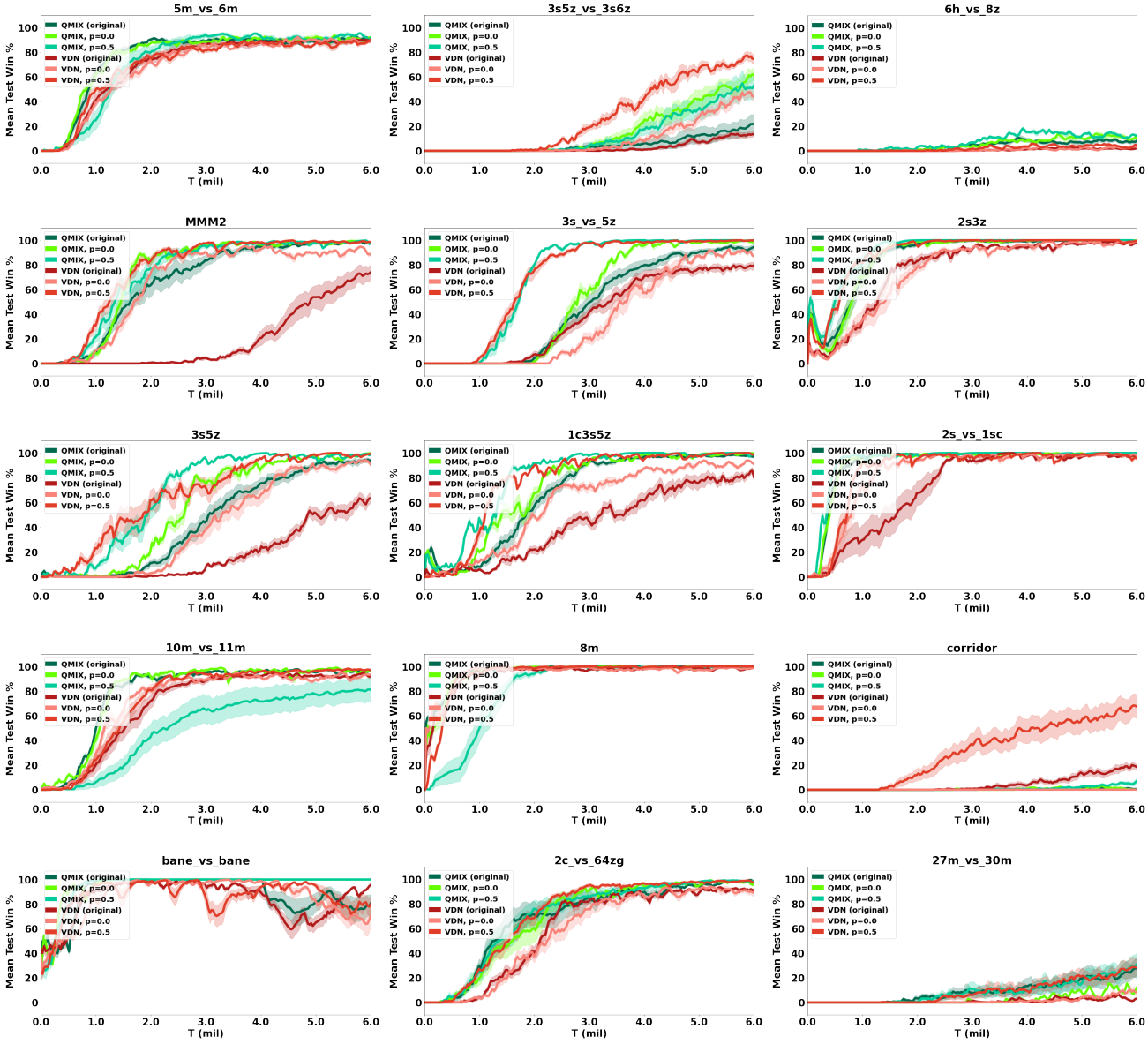

Figure 13. Comparison between VDN and QMIX baselines with original positive reward function and modified reward function with $p \in \{0.0, 0.5\}$ on SMAC maps.