A Value-based methods in a cooperative setting

Value-based methods consist of learning a Q function. Formally, in SARL, this corresponds to learning $Q^{\pi^*}(s,u) = \max_\pi Q^\pi(s,u)\ \forall (s,u) \in \mathcal{S} \times \mathcal{U}$. The optimal policy is a greedy selection: $u^* = \text{argmax}_u Q^{\pi^*}(s,u)$. Q-learning [Watkins and Dayan, 1992] showed that it is possible, when interacting with MDPs, to learn the exact Q functions using simple update rules depending only on the information collected by the agent. However, when $\mathcal{S} \times \mathcal{U}$ is very large or continuous, these methods can become intractable and function approximators are used to model the Q function. A neural network, parametrised by θ, can be used to learn a Q function approximation by minimising the objective defined in Equation 1 where B is the experience replay and θ' parametrises a target network.

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1}\rangle \sim B}\left[\left(r_t + \gamma \max_{u \in \mathcal{U}} Q(s_{t+1}, u; \theta') - Q(s_t, u_t; \theta)\right)^2\right] \tag{1}$$

The experience replay stores transitions $\langle s_t, u_t, r_t, s_{t+1}\rangle$ and the target network is a copy of θ that is periodically updated. In DQN [Mnih et al., 2015], θ parametrises a convolutional network while in DRQN [Hausknecht and Stone, 2015], θ parametrises a recurrent network (RNN), which has been shown to achieve better performance in partially observable environments. When θ is a recurrent network, B stores sequences of contiguous transition as the update of recurrent networks is performed on sequences.

When considering value-based methods in a Dec-POMDP, one possible method to consider is Independent Q-Learning (IQL) [Tan, 1993]. With IQL, agents independently learn their Q function, as in SARL, without considering the existence of other learning agents in their environment. One problem with IQL is that agents must select actions which maximise $Q(s_t, \mathbf{u_t})$ while ignoring, at any time, actions taken by other agents.

This is where CTDE becomes useful. It is possible to approximate $Q(s_t, \mathbf{u_t})$ as a factorisation of individual Q_a functions during training such that $\mathbf{u_t}$ maximises both the joint and the individual Q_a functions. To ensure this, individual Q_a functions must satisfy the Individual-Global-Max condition (IGM)[Son et al., 2019] presented in Equation 2.

$$\text{argmax}_{\mathbf{u_t}}\, Q(s_t, \mathbf{u_t}) = \bigcup_a \text{argmax}_{u_t^a}\, Q_a(s_t, u_t^a) \tag{2}$$

A.1 QMIX

QMIX [Rashid et al., 2018] is a CTDE method where the factorisation of $Q(s_t, \mathbf{u_t})$, denoted as $Q_{mix}(s_t, \mathbf{u_t})$, is performed as a function of the individual Q_a functions and the state during training. It is defined in Equation 3.

$$Q_{mix} = \text{Mixer}\left(Q_{a_1}(s_t, u_t^{a_1}), .., Q_{a_n}(s_t, u_t^{a_n}), s_t\right) \tag{3}$$

The mixer satisfies IGM by enforcing $\frac{\partial Q_{mix}(s_t, \mathbf{u_t})}{\partial Q_a(s_t, u_t^a)} \geq 0\ \forall a \in \{a_1, .., a_n\}$ by constraining a hypernetwork [Ha et al., 2016] to produce positive weights in order to factorise $Q(s_t, \mathbf{u_t})$. Formally, this is defined by a hypernetwork $h_p(.) : \mathcal{S} \to \mathbb{R}^{|\phi|+}$ which takes the state s_t as input and computes the strictly positive parameters[2] ϕ of a main network $h_m(.)$. This main network takes as input all individual Q_a to compute Q_{mix} with the positive weights and the offsets defined by ϕ. Together, $h_p(.)$ and $h_m(.)$ defines the mixer such that $h_m(.) : \mathbb{R}^n \times \phi \to \mathbb{R}$ and $Q_{mix}(s_t, \mathbf{u_t}) = h_m\left(Q_{a_1}(), .., Q_{a_n}(), h_p(s_t)\right)$. QMIX architecture is presented in Figure 4a.

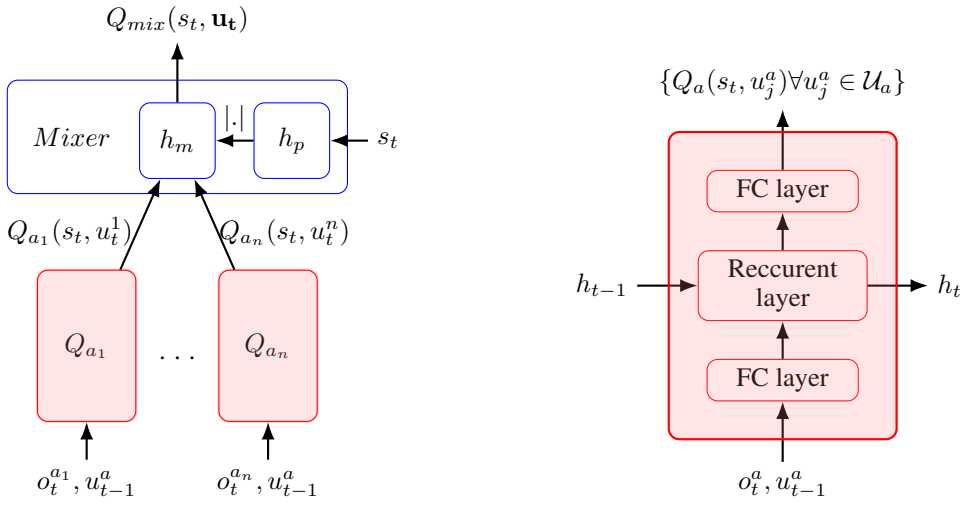
The monotonicity of Q_{mix} with respect to the individual Q_a functions is satisfied because a neural network comprised of monotonic functions (h_m) and strictly positive weights (h_p) is monotonic with respect to its inputs (Q_a). Because of partial observability, individual Q_a networks are RNNs made of GRU [Chung et al., 2014]. The optimisation procedure follows the same principles of the DQN algorithm and the loss is applied to $Q_{mix}(s_t, \mathbf{u_t})$ and defined in Equation 4.

[2] To be exact, the offsets defined by $h_p()$ are not constrained to be positive, only the weights.

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, \mathbf{u_t}, r_t, s_{t+1} \rangle \sim B} \left[\left(r_t + \gamma \max_{\mathbf{u} \in \mathcal{U}} Q_{mix}(s_{t+1}, \mathbf{u}; \theta') - Q_{mix}(s_t, \mathbf{u_t}; \theta) \right)^2 \right] \quad (4)$$

Since this method trains recurrent networks, the replay buffer does not store isolated transitions $\langle s_t, \mathbf{u_t}, r_t, s_{t+1} \rangle$ but instead stores sequences of contiguous transitions. Individual Q_a networks are copied as well was as the mixer to produce target networks represented by θ'.

In QMIX implementation, as in QVMix, individual Q_a architecture follows the architecture presented in Figure 4b and takes as input the previous action in addition to the observation. Note that the hidden states of the recurrent network are represented by h_t and their objective is to encode the agent's history τ_t.



(a) QMIX architecture. Individual Q_a are represented in red and the mixing network is represented in blue.

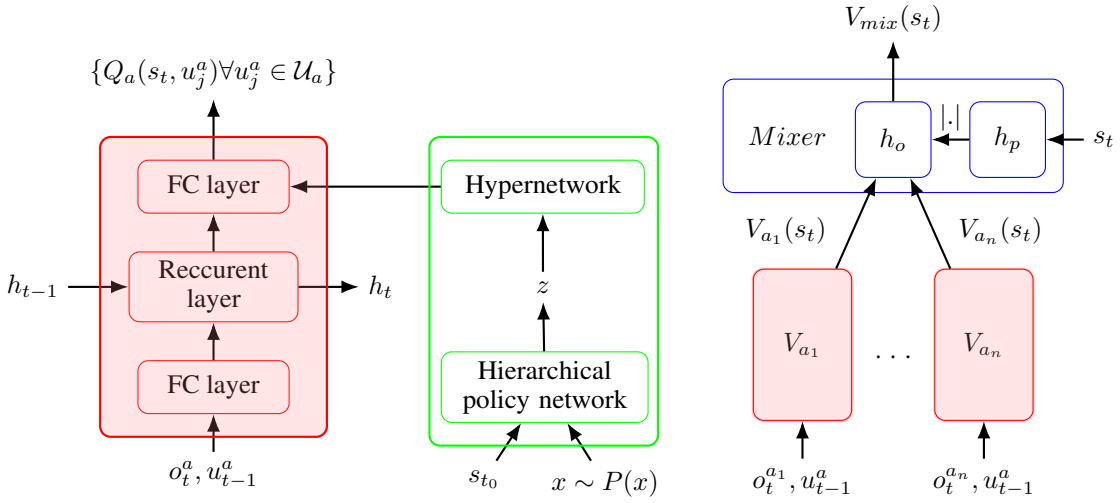
(b) Individual Q_a network implementation in QMIX and QVMix. The action space size defines the number of outputs of each agent network.

Figure 4: Details of the QMIX and QVMix architecture.

A.2 MAVEN

Mahajan et al. [2019] defined the class of state-joint-action value functions that cannot be represented by QMIX due to its monotonicity constraint. They demonstrated the existence of payoff matrices in an n-player game with more than three actions per agent for which QMIX learns a suboptimal policy for any training duration, and for epsilon greedy and uniform exploration. To tackle this problem, in the former QMIX architecture, they added a latent space that conditions individual Q_a function networks with the objective of influencing agent behaviour. This allows one to learn an ensemble of approximations and therefore an ensemble of policies to improve the exploration capabilities. The latent variable is the input of a hypernetwork, such as h_p in QMIX, that computes parameters for the fully connected layer linking recurrent cells to outputs in the individual Q_a networks. This latent variable z is generated per episode and by a hierarchical policy network, taking as input the initial state of the environment together with a random variable (typically discrete and sampled from a uniform distribution). The latent variable maps the different learnt strategies and the goal of the hierarchical policy network is to select the best strategy based on the initial state s_0 which is hypothetically known at testing. The architecture of the individual Q_a network of MAVEN is represented in Figure 5a.

MAVEN's network objective function comprises three parts. To optimise the two hypernetworks (the mixer and the latent space hypernetwork) and the individual recurrent networks, a part of the objective is the loss of QMIX defined in Equation 4. This loss is computed by fixing the hierarchical policy network and therefore the latent variable z. To optimise the hierarchical policy network, any policy optimisation, such as policy gradient [Sutton et al., 1999], computed with the total sum of



(a) MAVEN modification of the individual Q_a network.

(b) Architecture of the V_{mix} network.

Figure 5: Details of the MAVEN and QVMix architecture.

rewards per episode can be used. This second objective is computed by fixing both hypernetworks and the individual networks. To enforce that different values of z imply different behaviours, a mutual information loss between the latent variable and consecutive transitions is added as the third part of the objective. This third part of the objective requires the introduction of a variational distribution and for further details on the MAVEN optimisation procedure and especially on the construction of the mutual information objective, we refer the reader to Mahajan et al. [2019].

A.3 QVMix

QVMix [Leroy et al., 2020] is an extension of the Deep Quality-Value (DQV) family of algorithms [Sabatelli et al., 2018, 2020] to the cooperative MARL setting. The principle of DQV is to learn both the Q function, $Q(s, u; \theta)$, and the V function, $V(s; \phi)$, simultaneously.

In SARL, following the principles of DQN, both networks are trained with the losses defined in Equations 5 and 6.

$$\mathcal{L}(\phi) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1} \rangle \sim B} \left[\left(r_t + \gamma V(s_{t+1}; \phi') - V(s_t; \phi) \right)^2 \right] \tag{5}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, u_t, r_t, s_{t+1} \rangle \sim B} \left[\left(r_t + \gamma V(s_{t+1}; \phi') - Q(s_t, u_t; \theta) \right)^2 \right] \tag{6}$$

In QVMix, the V network is updated with the loss defined in Eq. 5 and has the same architecture as the Q network of QMIX, except that actions are not considered. The architecture is presented in Figure 5b. The Q_{mix} network remains the same as in QMIX but is now updated with the loss defined in Eq. 6. Again, B stores sequences of contiguous transitions instead of single transitions to train recurrent networks.

B Elo score

The purpose of the Elo rating system [Elo, 1978] is to assign each player of a population with a rating R to rank them. From these ratings, one can compute the probability that a player will win when facing another one. Let R_A and R_B be the ELO scores of player A and B, respectively. In such a context, the probability that player A (B) wins over player is B (A) is computed using Equation 7 (8) given below.

$$E_A = \frac{10^{R_A/400}}{10^{R_A/400} + 10^{R_B/400}} \tag{7}$$

$$E_B = \frac{10^{R_B/400}}{10^{R_A/400} + 10^{R_B/400}} \tag{8}$$

One can see that $E_A + E_B = 1$. The number 400 can be considered as a parameter. It determines that if the Elo score of a player A is 400 points above that of B, it has a ten-times greater chance of defeating B. In order to update the rating of player A after a game, we take into account its score S_A which is equal to 1 for a win, 0 for a loss and 0.5 for a draw. The updated score R'_A is defined in Equation 9 where cst is a constant that defines the maximum possible update of the Elo score. Typically, cst is 32 but for our experiments, we set it to 10 to decrease the amplitude of oscillations in the Elo score during tests.

$$R'_A = R_A + cst * (S_A - E_A) \tag{9}$$

C Competitive SMAC

(a) 3m (b) 3s5z

Figure 6: Overview of the 3m and 3s5z SMAC maps seen in StarCraft.

In SMAC, agents have partial observability defined by a sight range, a surrounding circle inside which they can observe allies and enemies and their shooting range is smaller than the sight range. There are different components in the observation of the state. An agent observes information about itself: its remaining hit points and shield points, its relative position with respect to the centre of the map and four Booleans representing the direction it can move in (NSWE). It also observes information about other agents that are within its sight range: the relative distance, relative x, relative y and the remaining hit points and shield points. If the other agent is an ally, it also observes the last action performed by the allied agent. If the other agent is an enemy, it observes if the enemy agent is within its shooting range.

In the $3m$ map, six marines compete in teams of three. A marine has 45 hit points and shoots at range, inflicting 6 damage points to an opponent for each attack. In the $3s5z$ map, six stalkers and ten zealots compete in teams of eight. Both units have shield points in addition to hit points. A shield receives a different amount of damage and regenerates over time if the unit is not attacked for a given period of time. A stalker has 80 hit points and 80 shield points. It shoots at range, inflicting 13 damage points to the shield, 12 damage points to a zealot's hit points and 17 to a stalker's hit points. A zealot has 100 hits points and 50 shield points. It attacks in melee and inflicts 16 damage points to the shield and 14 damage points to the hit points of a zealot or a stalker. On both maps, there are two possible starting positions for the teams (see Figure 6) and agents initially do not see their opponents.

Within the both maps, the agent has the choice of eight actions: do nothing, move in one of four directions (NSWE) or attack one of its three opponents. Some actions are forbidden in SMAC, such as an attack action if the opponent is not within shooting range. Therefore, before choosing an action, agents must consider which ones are available.

At each timestep, the agent receives a zero or positive reward, common to each agent of the same team. This reward is the sum of a zero or positive reward for the damage dealt, a positive reward if an enemy unit's hit points reach zero, and a positive reward if all enemy units are defeated. Maximising the reward forces the team to neutralise every unit of the opposing team.

D Training parameters

Learning parameters of the three methods were determined by default configurations provided by their different authors. They are the same as the ones used in QVMix implementation [Leroy et al., 2020]. We hereafter provide a description of some of these training parameters.

Individual networks are 64 cells GRU enclosed with fully-connected layers (see Fig 4b). The mixer network is the same as in [Rashid et al., 2018] with an embedded size of 32. The individual and mixer networks are the same for the three methods. We used the default parameters of MAVEN policy networks provided by [Mahajan et al., 2019]. For QVMix, the V network is a copy of the QMIX network with only one output for each V network.

For each learning scenario, networks are updated regardless of how episodes have been generated. Networks are updated from a replay buffer that collects the 5000 latest played episodes and 32 of them are sampled from it to update the network. The network update is performed every eight episodes in the $3m$ map and every episode in the $3s5z$ map. The difference is justified by the desire to increase the number of network updates for $3s5$ to improve performance, especially against the heuristic. The epsilon greedy exploration starts with an epsilon equal to 1 decreasing linearly to 0.05 during 2 million timesteps. This is perhaps the main difference with respect to the provided parameters that decreases the epsilon only during 0.5 million timesteps. The discount factor is $\gamma = 0.99$ and the learning rate is 0.0005. Target networks are updated every 200 episodes. We refer the reader to [Mahajan et al., 2019] for further parameter definitions for MAVEN optimisation.

E Training time

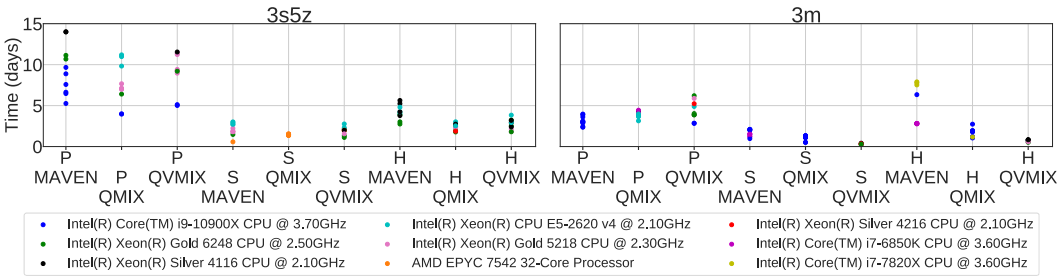
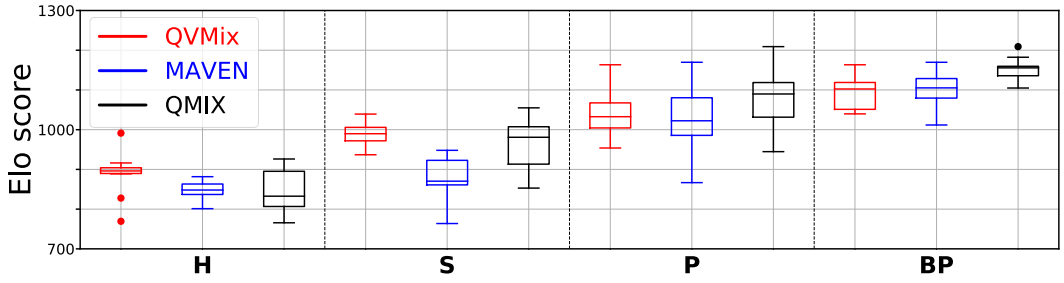


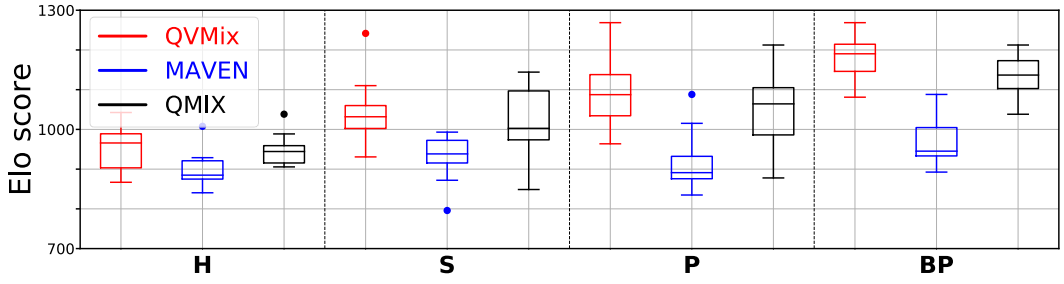
Figure 7: Training duration, in days, for the different training scenarios tested in this paper. The different colours represent the different CPUs that have been used to perform the experiments.

Experiments were performed with CPUs only because small recurrent networks do not arguably benefit from GPU. We had access to several types of computer with different numbers of CPUs accessible at the same time. Training times for each experiment performed are presented in Figure 7. With all these different hardware configurations, it is not possible to rigorously compare the time efficiency of the experiments. However, it is to present the time complexity. As explained in Section 3, training in self-play requires five-times fewer environment timesteps than training within a population, but also five-times fewer network updates. Furthermore, when training a population, networks are updated sequentially, which also increases the time. Finally, SC2 processes are prone to bugs and therefore sometimes need to be restarted. As the actions of all agents in the different running environments are performed simultaneously, these restarts are time-consuming operations as the processes have to wait for the faulty one.

F Additional Elo score boxplots



(a) $3m$



(b) $3s5z$

Figure 8: Elo score box plots of two test populations, in $3m$ at the top and in $3s5z$ at the bottom, composed of teams trained using three methods and three learning scenarios. The training method is either QVMix (red), MAVEN (blue) or QMIX (black). Box plots represent the distribution of the Elo scores of teams trained either against the heuristic (**H**), in self-play (**S**), within a population (**P**) or the best of each population (**BP**). Box plots present the median, the first quantile ($Q1$) and the third quantile ($Q3$). The reach of whiskers is defined by $1.7 * (Q3 - Q1)$.

Boxplots of the test populations composed of all methods without the heuristic for both maps are presented in Figure 8.