RL Assignment 3: Pacman Capture the Flag - Final report

Group 14 : Michaela Hanajíková, Nicola Maritan, Maria Sgaravatto

April 26, 2025

1 QMIX implementation

In the first part of the assignment, we were tasked with implementing Independent Q-Learning (IQL) and QMIX to test on the `smallCapture` map. We began by implementing IQL, where each agent learns independently by updating its own Q-values based on its observations and actions. The initial results were promising, but we wanted to see if this performance could be further increased.

Following the implementation of IQL, we explored the use of Value-Decomposition Networks (VDN) to enhance performance. The rationale behind this approach was that by first implementing VDN and ensuring its correct functionality, it would be easier to extend it to QMIX by introducing a QMixer and a hypernetwork. VDN extends the Q-learning framework by incorporating inter-agent dependencies, decomposing the joint action-value function into a summation of agent-specific Q-values. This approach enables the system to effectively learn coordinated behaviors among agents.

Subsequently, after successfully implementing VDN, we proceeded to integrate QMIX. QMIX builds upon VDN by learning a more flexible mixing function to combine individual Q-values, instead of relying solely on summation. The performance comparison between the initial IQL approach and QMIX is presented in Section 1.2.

1.1 Improvement over time

The QMIX agents start with relatively low team rewards initially, with the scores being near zero, with significant fluctuations. Over time, the team rewards of QMIX agents show rapid improvement, especially within the first 20,000 steps. The QMIX agents also display a clear trend of improvement in their scores over training time. After reaching a higher reward level (around 35–40), the QMIX agents stabilize and maintain consistently high team rewards for the remainder of training. Similarly, the the scores steadily rise and stabilize at around 10–12 after the initial 20,000 steps. Occasional fluctuations can be observed, but the overall performance remains relatively stable, which indicates effective learning and convergence.

1.2 Performance comparison

As described in the assignment, we compared the IQL and QMIX implementation on the `smallCapture` layout against `randomTeam` using the same hyperparameters.

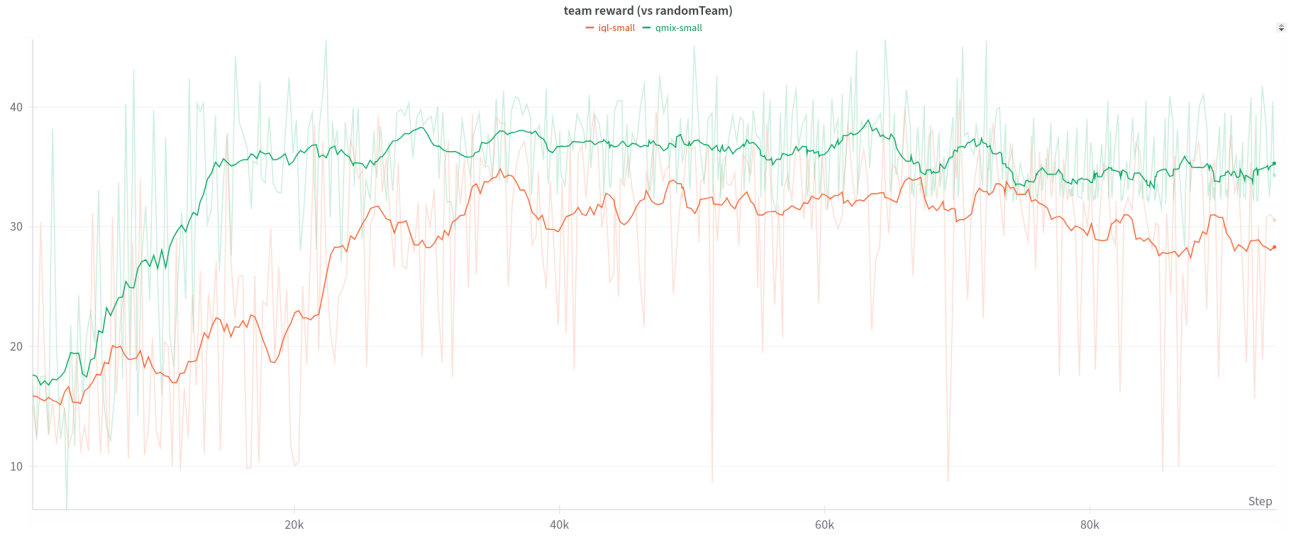

Figure 1: Team reward of IQL and QMIX on `smallCapture` against `randomTeam`.

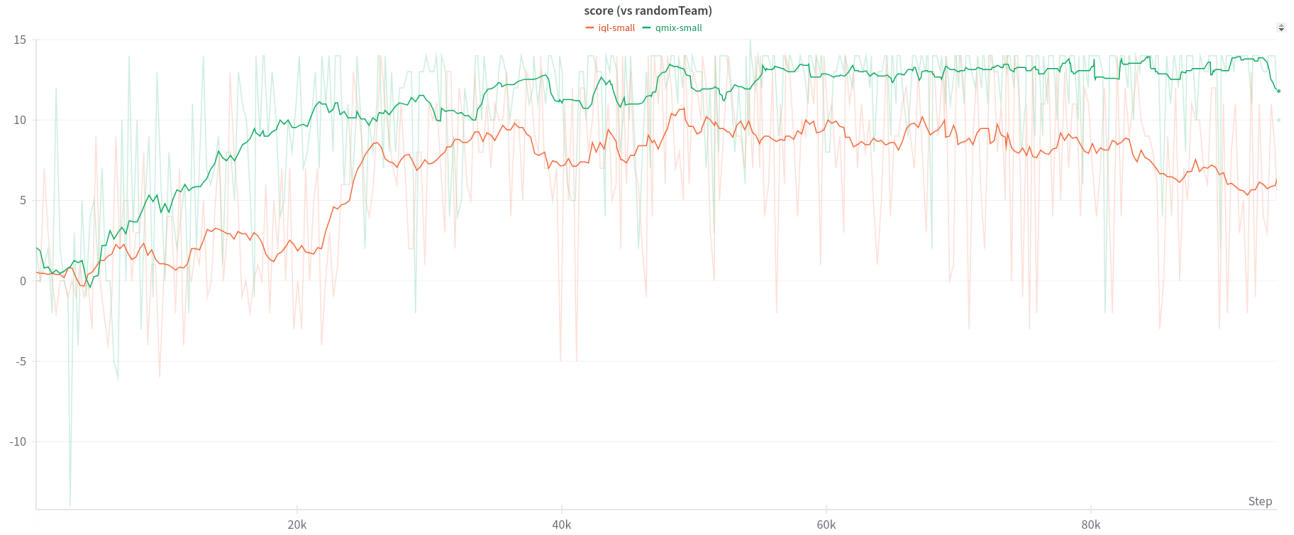

Figure 2: Score of IQL and QMIX on `smallCapture` against `randomTeam`.

QMIX generally produced better results in terms of performance, as reflected in the overall higher team reward and score (Figure 1 and Figure 2), surpassing the initial IQL. As expected, IQL exhibited slower learning rates while QMIX showed progressively better results. Additionally, the cooperation between agents in the Qmix implementation was more robust, and the team was able to efficiently steal food from the opposing side while defending its own. This is further described in the following section.

1.3 Different roles within a team

The primary difference observed within a team pertained to the positioning of agents on the field. Specifically, the agents learned to occupy different vertical sides of the field, enabling more effective coverage of each area. In certain scenarios, such as when competing against more advanced teams, we observed a complementary dynamic in which one agent adopted an attacking role, while the other shifted to a defensive position.

1.4 Reflection

In conclusion, our sequential implementation of IQL, VDN, and QMIX provided valuable insights into how these algorithms scale with increasing levels of agent interaction and coordination. We observed consistent performance improvements from IQL to QMIX, enabling agents to perform effectively against the `randomTeam`. However, testing the final agents from this phase against other teams revealed several weaknesses. These findings

informed the subsequent steps of our work, where QMIX was utilized as a baseline for further algorithmic improvements.

2 Improving the training strategy

In this section, we detail the improvements made to our training strategy and evaluate the performance of our algorithm against stronger teams. The rewards and scores obtained during training against all baseline teams are presented in Figures 3 to 8.

2.1 Challenges in training multi-agent systems

In Section 1, we trained our agents against an opponent team that selected random actions at each time step. Since the QMIX algorithm we implemented models the opponent team as part of the environment, the resulting policies are inherently adapted to this specific opponent behavior. Consequently, the trained agents are unlikely to generalize effectively when faced with new opponents that employ different strategies or play in a more structured and strategic manner.

To train two agents capable of competing against various strategies, we propose exposing them to a range of opponents with different tactics, as outlined in the assignment. The goal is to enable the agents to adapt to diverse strategies. To achieve this, we will enhance the training process by randomly selecting an opponent team at each episode.

During the initial 100 episodes, the agents are trained against the `randomTeam`. This stage aims to familiarize the agents with the fundamental rules of the game and the primary mechanisms for accumulating rewards. Since the `randomTeam` lacks any strategic behavior, the agent predominantly learns aggressive strategies focused on maximizing capsule collection from the outset. However, prolonged training against the `randomTeam` risks overfitting to this aggressive behavior, which is insufficient for competing against more strategic opponents. To mitigate this, training against the `randomTeam` is limited to 100 episodes.

Subsequently, the agents are trained exclusively via self-play for 100 episodes. This transition is designed to enable the agent to counter its previously learned aggressive strategies by developing defensive capabilities while simultaneously exploring more sophisticated offensive tactics. Self-play provides the agent with a dynamic and evolving opponent as its own strategies adapt over time. To ensure a stable learning environment, the self-play agents were updated every 10 episodes instead of after every episode. Self-play training begins by resetting the ϵ of the ϵ-greedy action policy to 0.5. Each team is assigned its own ϵ value, which decays progressively over the course of training. This approach ensures that agents independently explore strategies against each team during training. Notably, we do not utilize separate replay buffers for each team. This decision is grounded in the fact that QMIX, and off-policy algorithms like DQN in general, can leverage all past experiences, including those involving different types of opponents, for continuous learning. However, as detailed in Section 2.4, we incorporate modifications to the replay buffer to prioritize learning from the most significant transitions collected during training.

Following the initial phase, we incorporate the weakest among the smart teams into the training process, specifically the `baselineTeam` and `MCTSTeam`, as identified by intermediate tournament results. For these teams, the ϵ parameter is reset to 0.5, and during each episode, a team is sampled uniformly from the self-play pool, `MCTSTeam`, and `baselineTeam`. After 300 episodes, stronger teams, including `AstarTeam`, `approxQTeam`, and `heuristicTeam`, are introduced. Their corresponding ϵ values are also reset to 0.5, and subsequent team sampling is performed uniformly across the entire set of teams.

An alternative approach considered was to gradually introduce stronger teams by sampling from a non-uniform distribution. In this setup, sampling probabilities were inversely proportional to team strength initially and were adjusted over time to converge to a uniform distribution, controlled by a temperature parameter. While promising, this method added complexity by introducing additional parameters to the training algorithm. Consequently, we opted for the simpler uniform sampling strategy described above.

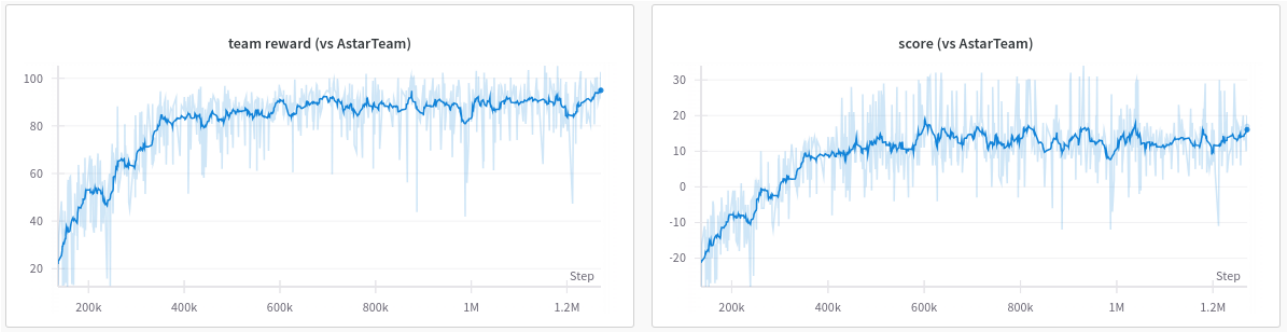

Figure 3: QMIX with improved training strategy team reward and score on `bloxCapture` against `AstarTeam`.

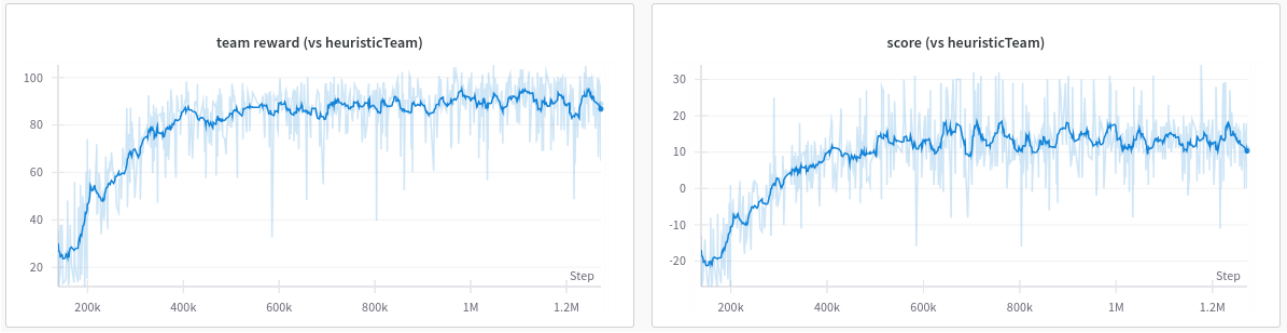

Figure 4: QMIX with improved training strategy team reward and score on `bloxCapture` against `heuristicTeam`.

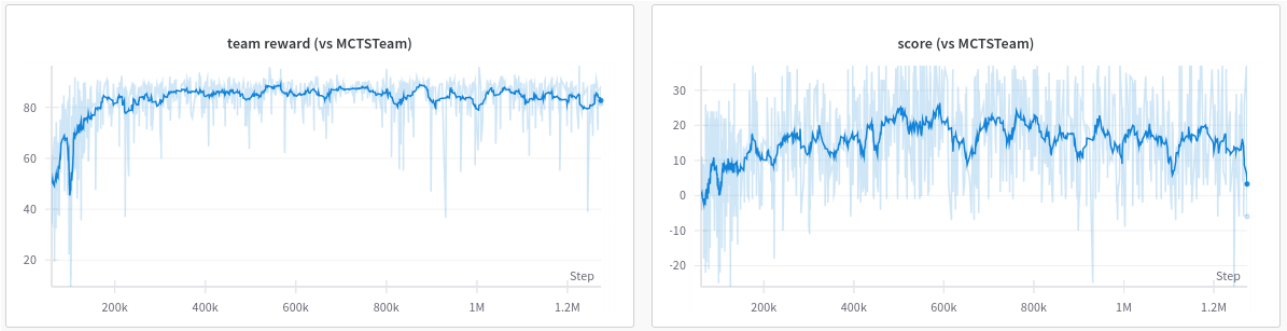

Figure 5: QMIX with improved training strategy team reward and score on `bloxCapture` against `MCTSTeam`.

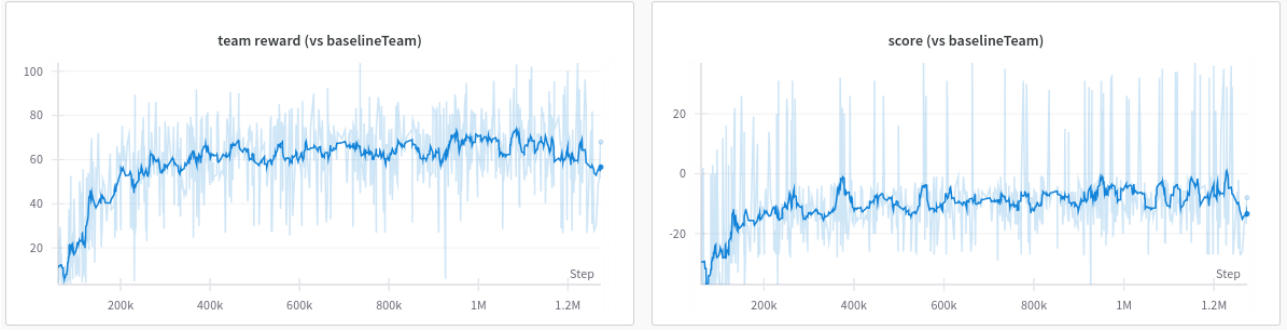

Figure 6: QMIX with improved training strategy team reward and score on `bloxCapture` against `baselineTeam`.

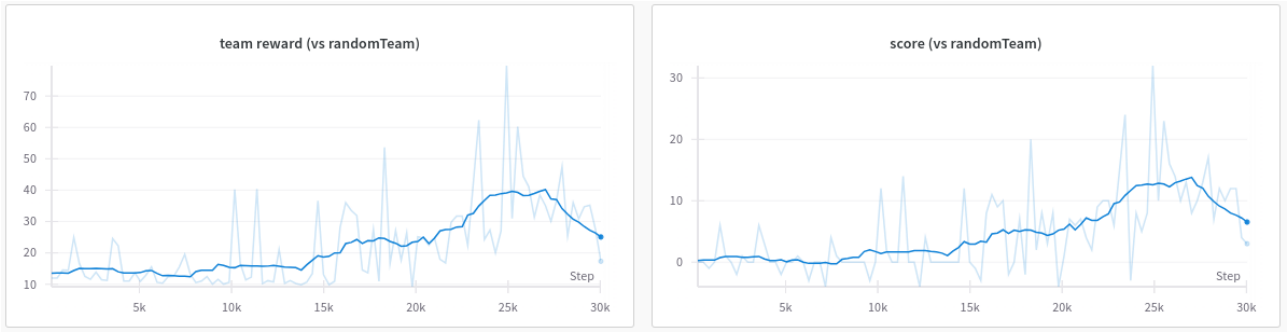

Figure 7: QMIX with improved training strategy team reward and score on `bloxCapture` against `randomTeam`.

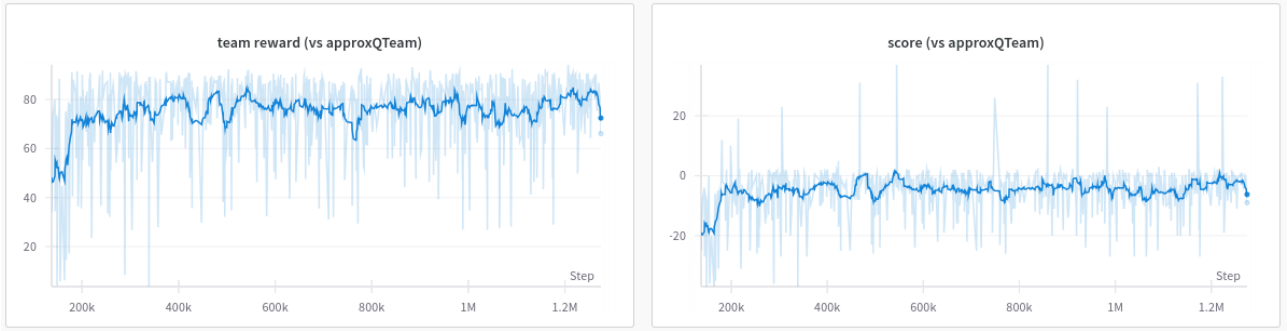

Figure 8: QMIX with improved training strategy team reward and score on `bloxCapture` against `approxQTeam`.

2.2 Stability of learning

To address the instability observed during training with hard updates, where the loss function exhibited a significant increase over time (rising from approximately 10 to 150), we implemented a soft update mechanism. This approach allowed for more stable training dynamics and helped maintain a consistent loss throughout the training process.

Additionally, we employed gradient clipping as another stabilization technique. By enforcing a maximum norm of the gradient at each training step, we prevented excessively large updates that could destabilize the learning process. Specifically, we capped the gradient norm at a value of 10, which contributed to improved training stability and more controlled optimization behavior.

2.3 Policy fine-tuning

While gradient clipping helps stabilize training, we also implemented a learning rate schedule to further enhance policy optimization. Specifically, we employed a decreasing step learning rate schedule with a minimum value of 10^{-5}. This approach ensures that the learning rate starts at a sufficiently high value to enable effective exploration of the parameter space during the initial phases of training and gradually reduces over time to facilitate convergence to a more optimal policy as training progresses.

2.4 Replay Buffer Usage

A critical component of QMIX, and DQN-based methods in general, is the replay buffer, which is essential for storing diverse experiences. The primary techniques for ensuring diversity in the replay buffer are maintaining a high exploration rate during the initial training phase and utilizing a large buffer size.

2.4.1 Prioritized replay buffer

We upgraded our training strategy to use a prioritized replay buffer. Unlike a standard replay buffer, which samples transitions uniformly at random, a prioritized replay buffer assigns higher sampling probabilities to transitions with greater learning potential, measured by their temporal-difference error. This approach accelerates learning by focusing on more informative experiences, thereby improving sample efficiency.

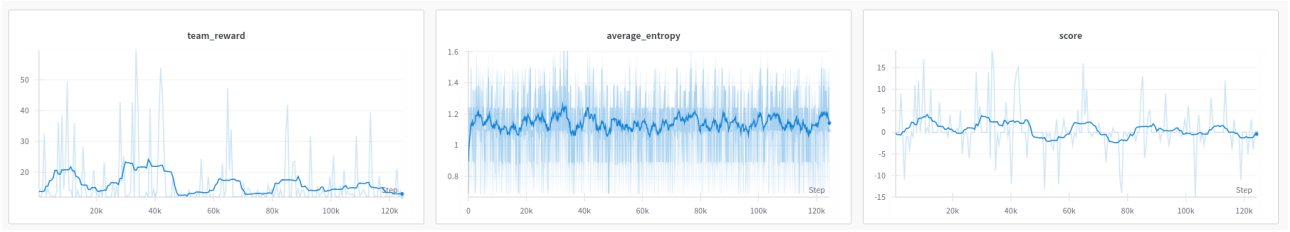

Figure 9: Score, team reward and action distribution entropy using Boltzmann-Exploration on `bloxCapture` against the `randomTeam`.

2.4.2 Global state representation

A larger replay buffer size enables the storage of more experiences, including both current and future observations, as well as global states. However, this approach can become memory-intensive if not managed efficiently. Initially, global states were generated by stacking the local observations of all four agents. For the `bloxCapture` layout, this resulted in each replay buffer entry containing $4 \times 8 \times 20 \times 20$ elements. Upon further analysis, we observed that many layers of the local observations were identical across agents. To optimize memory usage, we modified the global state construction process to avoid redundancy. The new global state is created from the local observations as follows:

- Walls: The wall data, represented as the first channel of the local observations, is identical for all agents.
- A separate channel is allocated for each agent's position and the number of capsules it is carrying. Since each agent is only aware of how many capsules it is carrying, this information is incorporated into the global state by stacking the individual capsule counts from each agent's local observation.
- Blue Capsules: Blue capsule data, represented as a single channel, is identical across agents.
- Red Capsules: Red capsule data, similarly represented as a single channel, is shared across agents.
- Blue Food: The blue food layer is identical for all agents.
- Red Food: The red food layer is also shared across agents.

A global state constructed using this method contains only $9 \times 20 \times 20$ elements, which is approximately 0.281 times the size of the global state generated by the naive approach. In addition, this enables faster computations during training, leading to shorter overall training times.

2.5 Self-play performance

In Figure 10, we present the score and reward trends during training against self-play. Unsurprisingly, the score oscillates around 0. This outcome is expected, as the team competes against itself, resulting in victories or losses that are largely influenced by random factors (e.g., random actions caused by ϵ-greedy exploration) or by the delayed update of the self-play team. However, the team reward remains consistently high. This indicates that, despite an average score of 0, the teams are still able to accumulate rewards during gameplay by completing tasks such as collecting capsules and defending effectively.

2.6 Boltzmann-Exploration

During the development of this section, we also experimented with Boltzmann exploration as an alternative to ϵ-greedy. However, after some time, we did not observe a clear improvement in performance and, consequently, decided to continue using ϵ-greedy. In addition to monitoring reward and score, we tracked the entropy of the generated probability distribution over possible actions throughout training to assess the exploration status. We anticipated a gradual decrease in both the average entropy and entropy variance, which would indicate a shift in agent behavior from exploration to exploitation. However, this expected trend was not observed. We report our results on `bloxCapture` against the `randomTeam` in Figure 9.

3 Exploring Custom Algorithms

In this section, we examine three algorithmic approaches:

- Independent-PPO (I-PPO): A decentralized approach using Proximal Policy Optimization (PPO).
- VMIX-PPO: A hybrid method combining QMIX and PPO.

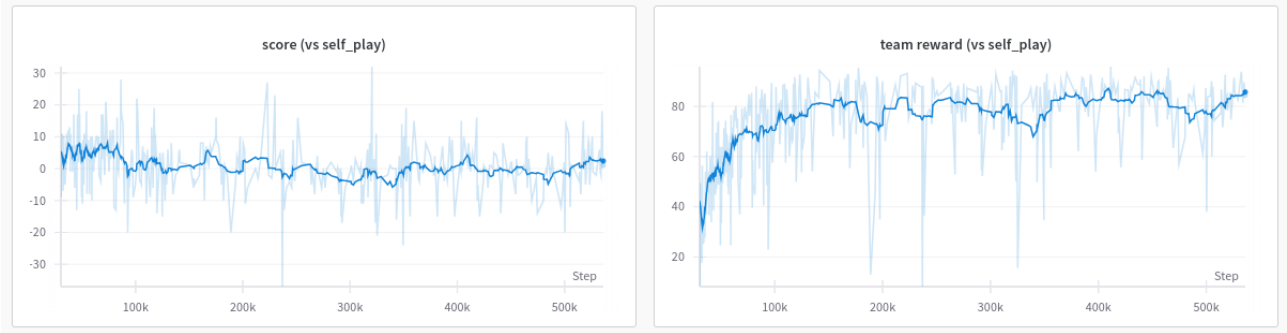

Figure 10: Self-play score and team reward on `bloxCapture`.

- Global-Value-PPO (GV-PPO): A PPO variant incorporating a global value function.

3.1 Independent-PPO

Independent PPO employs decentralized training and execution. Here, two agents independently optimize their rewards using the PPO algorithm without accounting for team rewards.

3.2 VMIX-PPO

VMIX-PPO is a hybrid approach integrating QMIX and PPO. Training is centralized, while execution remains decentralized. The value for each time step is calculated by aggregating individual agent values and incorporating the global state:

$$A_{\text{tot},t} = G_{\text{team},t} - V_{\text{tot},t}, \tag{1}$$

$$V_{\text{tot},t} = V_{\text{mixer}}\left(V_{\text{agent }1,t}\left(o_{1,t}\right), V_{\text{agent }3,t}\left(o_{3,t}\right), s_t\right), \tag{2}$$

where $o_{i,t}$ represents the observation of the i-th agent at time t, s_t is the global state, and $G_{\text{team},t}$ is the discounted cumulative team reward at time t. The idea is that, by employing this advantage formulation, the agents learn to optimize the team reward rather than their individual rewards. Similar to QMIX, this method assumes that increasing the value of an individual agent leads to an increase in total value. A hypernetwork generates the weights for the VMIXer, utilizing an absolute value activation function in the final layer.

3.3 Global-Value-PPO

In Global-Value-PPO, we use a global value function V_{global} in the advantage computation, i.e.,

$$A_{\text{tot},t} = G_{\text{team},t} - V_{\text{global},t}, \tag{3}$$

$$V_{\text{global},t} = V_{\text{global}}\left(s_t\right). \tag{4}$$

Similarly to Section 3.2, we use a value function during training that estimates the expected reward based on the global state, but we do not compute this by mixing individual agent values. Indeed, training is centralized, while execution remains decentralized. Ideally, by employing this advantage formulation, the agents learn to optimize the team reward rather than their individual rewards.

3.4 Results

We now present the results for the three methods on the `bloxCapture` layout. For these experiments, we used n-steps bootstrapping to compute the estimated returns.

In Figure 11 and Figure 12, we show the results with $n = 10$. It is evident that VMIX-PPO does not achieve the performance levels of GV-PPO and I-PPO. In particular, VMIX-PPO demonstrates the slowest performance increase in this setting. This outcome may be expected, as the method involves additional complexity due to learning the parameters associated with the mixing network. Furthermore, both VMIX-PPO and GV-PPO exhibit greater variance compared to I-PPO, which shows more stable training behavior. The final performances of GV-PPO and I-PPO are approximately the same.

In Figure 13 and Figure 14, we present the results with $n = 50$. Under this setting, VMIX-PPO matches the performance of the other two methods. The improvement compared to the previous experiment could be attributed to the increased number of samples available for bootstrapping, allowing VMIX-PPO to estimate

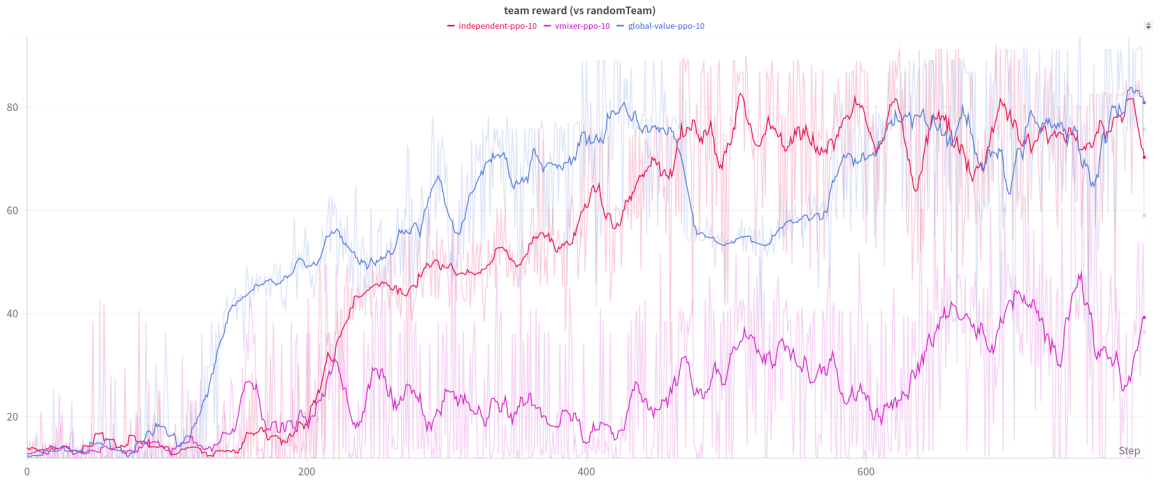

Figure 11: Team reward for I-PPO, VMIX-PPO and GV-PPO with $n = 10$.

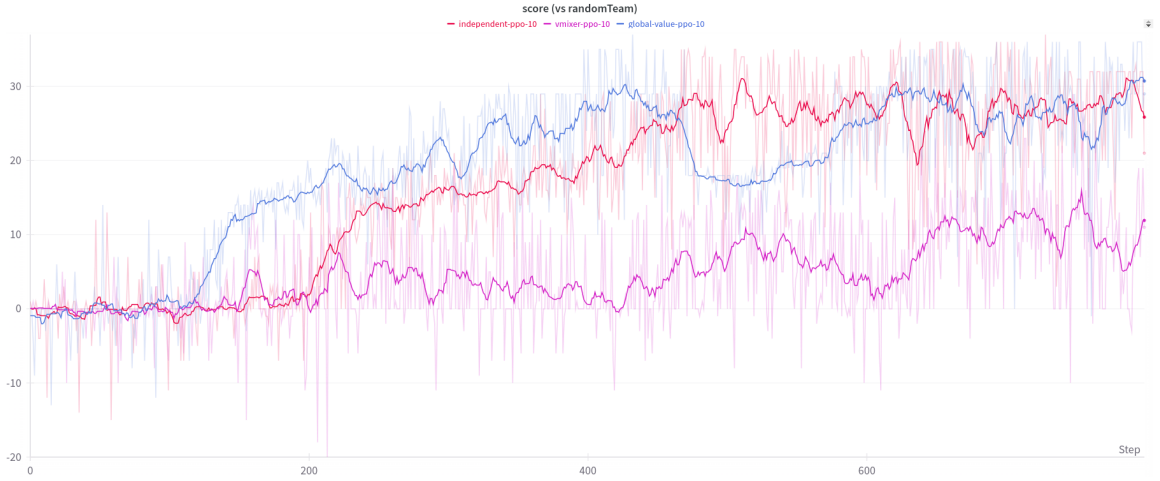

Figure 12: Score for I-PPO, VMIX-PPO and GV-PPO with $n = 10$.

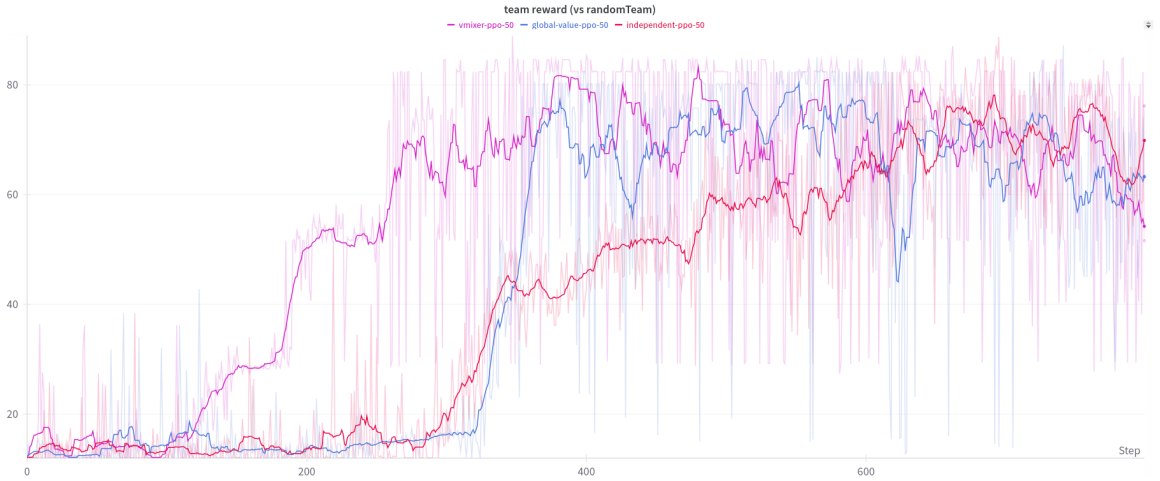

Figure 13: Team reward for I-PPO, VMIX-PPO and GV-PPO with $n = 50$.

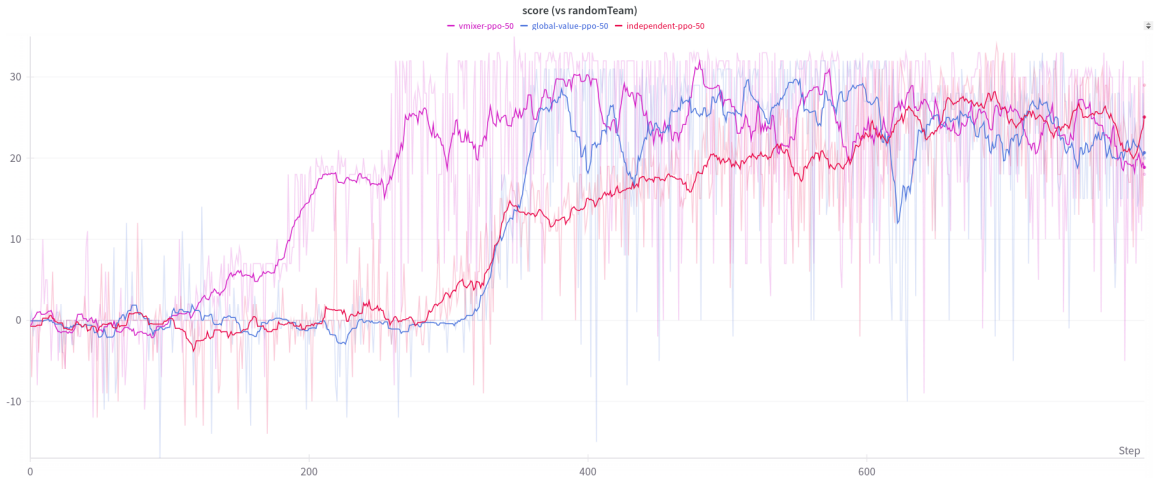

Figure 14: Score for I-PPO, VMIX-PPO and GV-PPO with $n = 50$.

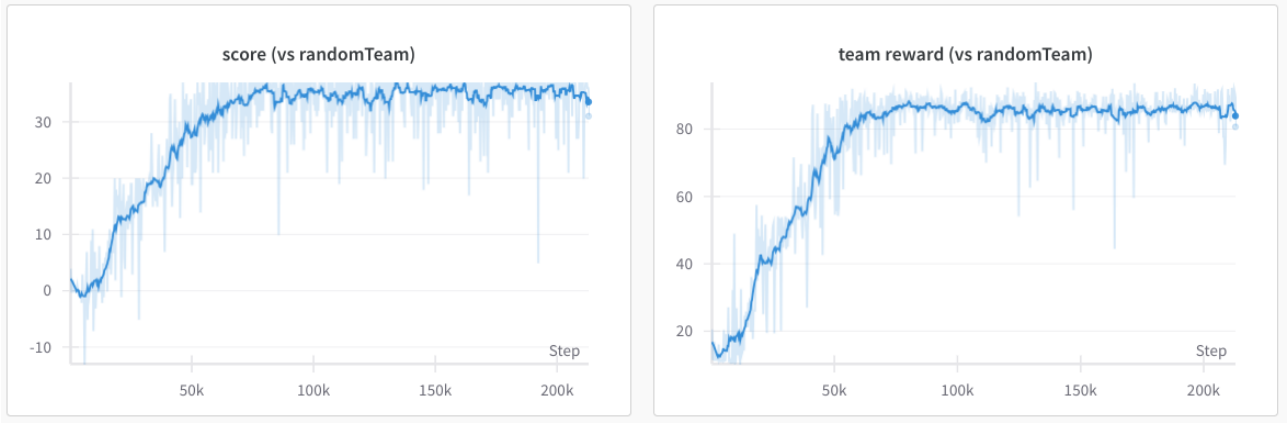

Figure 15: QMIX score and team reward on `bloxCapture` against `randomTeam`.

values as effectively as the other approaches. It could be that with $n = 10$, VMIX-PPO could achieve comparable performance given additional computational time. Once again, the I-PPO approach exhibits the most stable training across settings. The final performances of the three algorithms are approximately the same.

From the results, we can deduce that the two proposed algorithms, VMIX-PPO and GV-PPO, inherently exhibit higher variance compared to their independent counterpart, I-PPO. This increased variance arises from the challenge of computing a global value function for all agents, which is more complex than independently estimating value functions for each agent. While the idea of VMIX-PPO and GV-PPO is to optimize the team reward rather than individual rewards, this design introduces additional challenges that may require further tuning. For instance, hyperparameter adjustments, improvements in the architecture of the mixing or global value networks, or using more robust advantage estimation methods could help reduce variance and stabilize training.

3.5 Comparison with QMIX

In Figure 15 we report score and reward for QMIX against the `randomTeam` on the `bloxCapture` layout. By comparing these results with the proposed algorithm ones, we can clearly see that the improvement over time of QMIX is much more stable. However, it is important to notice that QMIX training takes many more training steps than the proposed methods.

3.6 Agent coordination

The only two algorithms that encourage coordination between the two agents are VMIX-PPO and GV-PPO. By incorporating a global value function during training, these methods guide the agents to select actions that maximize the team reward rather than individual rewards. In contrast, I-PPO does not facilitate coordination, as each agent independently focuses on maximizing its own reward without considering the overall team objective.

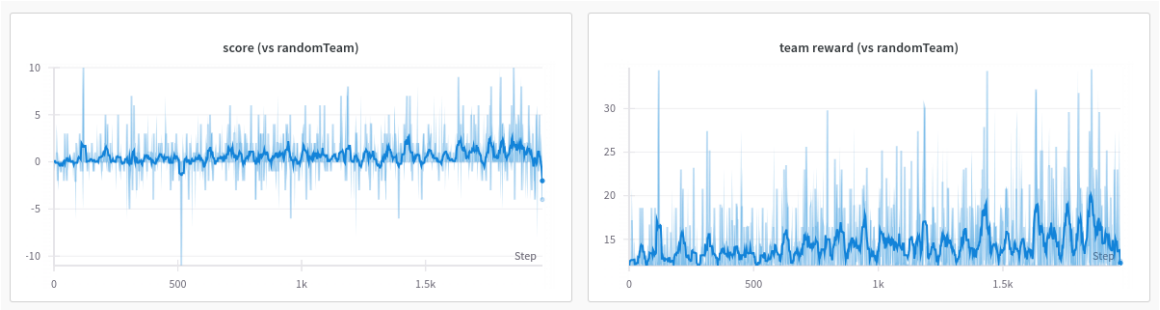

Figure 16: I-PPO score and team reward on random layouts against `randomTeam`.

3.7 Random layouts

We trained I-PPO with $n = 50$ on random layouts. Unfortunately, the algorithm showed only a slight increase in team reward over time and failed to achieve significant improvements. The results are shown in Figure 16. One potential approach to enhance performance would be to reduce the number of `ppo_epochs` to 1, minimizing overfitting to the current episode, and to increase the networks size and training episodes. However, due to time constraints, we were unable to test this configuration.

4 Additional algorithm

We also attempted to implement a Deep Recurrent Q-Network (DRQN) as an agent using the QMIX algorithm, following the approach described in the original paper. For the recurrent component, we employed a Gated Recurrent Unit (GRU). During training, episodes were stored in a replay buffer, from which sequences of a predefined length were randomly sampled. Additionally, the GRU hidden states were stored alongside the other transition elements. However, due to time constraints, we were unable to achieve a functional implementation that could learn effectively.