Ghent University

Faculty of Engineering and Architecture
Department of Information Technology

Multi-Agent Reinforcement Learning

Pacman Capture the Flag - Assignment 3

Enrico Saro
Francesco Pisacane
Simone Santin

Prof. Pieter Simoens

Reinforcement Learning

Academic Year 2025/26

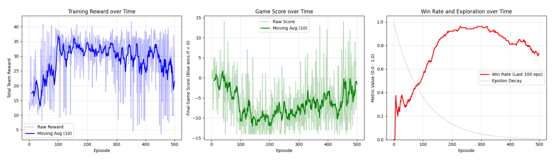
The comparative results discussed in this report rely on hyperparameter configurations chosen through manual tuning and limited experimentation. Given the vast search space and the computational cost of training, we could not guarantee the optimal configuration for every architecture. As advanced MARL algorithms are notoriously sensitive to hyperparameters, some performance disparities may stem from this suboptimal tuning. Consequently, our findings reflect the performance of these algorithms under the specific conditions we could feasibly test, rather than their theoretical maximum potential.

1 Part 1: QMix Implementation

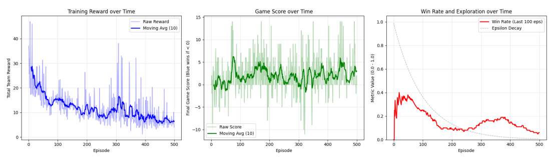
1.1 Reflection Questions

In this section, we evaluate the performance of our initial QMix implementation against the Independent Q-Learning (IQL) baseline. The agents were trained on two layouts: `smallCapture.lay` (a smaller and denser map) and `bloxCapture.lay` (the larger tournament map).

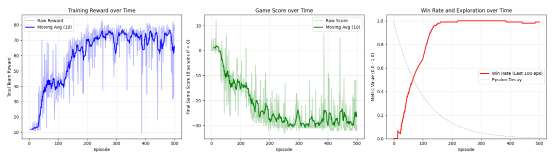
The comparative results of the training metrics (reward, game score, and win rate) are illustrated in Figure 1.



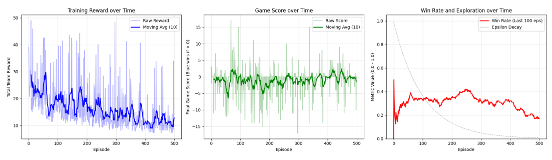
(a) IQL on *smallCapture*



(b) QMix on *smallCapture*



(c) IQL on *bloxCapture*



(d) QMix on *bloxCapture*

Figure 1: Training progress comparison between IQL and QMix on different layouts. The plots track Total Reward, Game Score, and win rate over episodes.

Based on the results shown in Figure 1, we answer the reflection questions below:

1. How do your QMix agents improve over time during the training?

On the larger map, *bloxCapture* (Figure 1d), the QMix agents struggle significantly to learn a robust policy. Unlike the steady improvement we hoped for, the win rate peaks at a modest ≈40% around episode 250, before slowly degrading towards ≈20% as the exploration rate decays. This indicates that the agents are unable to stabilize and maintain a winning strategy during the low-ϵ regime, when exploration is minimal and performance depends on the learned value estimates.

Similarly, on the *smallCapture* map (Figure 1b), the training exhibits severe instability. After an initial improvement where the win rate climbs to 40%, the performance collapses (catastrophic forgetting).

2. How does the performance of QMix compare to IQL?

Contrary to theoretical expectations that QMix should outperform IQL in cooperative tasks, our experiments show that IQL dramatically outperforms QMix in this specific setup.

- On *bloxCapture*, IQL reaches a win rate of nearly 100% (1.0) very quickly (Figure 1c), whereas QMix fails to even reach 50%.
- IQL agents achieve significantly higher final average reward ($\approx$70 vs. $\approx$15 for QMix).

We identified two primary reasons for this discrepancy.

First, the increased modeling complexity required by QMix can be detrimental in a relatively simple environment and when facing a random opponent. Estimating a joint value function through centralized components imposes a higher sample complexity compared to independent learners. As a consequence, the architecture typically requires more training episodes to converge to a good policy.

Second, and likely most importantly, our current QMix implementation uses a very shallow mixing network (single layer), which severely limits the mixer's expressivity. This likely hinders the mixer's ability to accurately estimate Q_{tot} from individual Q_a values on the complex map. By looking at the figure below, it becomes evident that deepening the QMix mixing networks allows the architecture to unlock the true potential of centralized training, even though the improvement is not fully appreciable in such a simple environment. In particular, the following results were obtained by implementing the original QMix architecture as described in the paper [1], where the mixing network is composed of a non-linear mixer with one hidden layer and its weights are generated by two-layer hypernetworks conditioned on the global state (processed via a CNN):

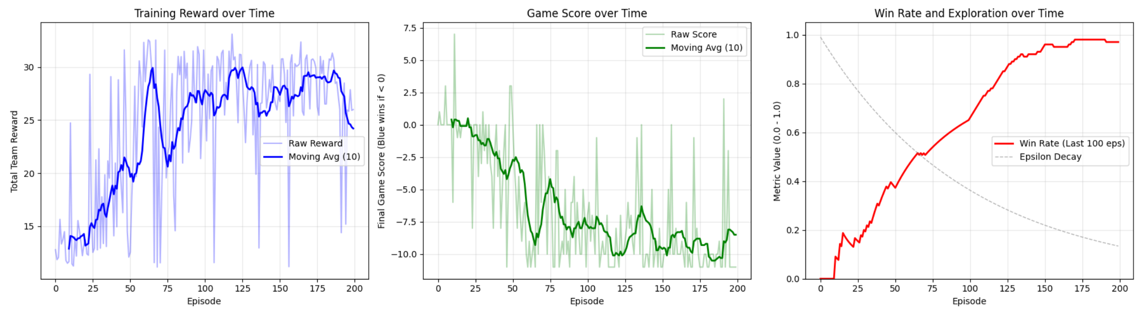


Figure 2: QMix training progress on bloxCapture against random agent, using enhanced architecture.

3. Do you observe different roles for the agents within a team?

By looking at the replays, it is clear that:

- IQL (Homogeneous Attackers): They don't seem to cooperate at all, adopting an "Attacker" role, rushing to consume food. This is also evident through the high total reward and fast convergence, and is to be expected since IQL agents maximize individual rewards.
- QMix (Confusion/Stagnation): The agents are not coordinating effectively but rather stagnating. The monotonicity constraint in our simple mixer might be forcing the agents into a local optimum where they play too passively to avoid negative rewards (being eaten), rather than actively cooperating to capture food.

4. What other reflection questions can you think of yourself?

- The Pacman CTF game might require non-monotonic value factorizations (e.g., a "sacrificial" play by Agent A to let Agent B score). If the optimal joint policy is not monotonic, which could be the case in CTF, the simplest version of QMix is theoretically unable to represent it and may therefore struggle. This limitation is a key motivation for improving this architecture and for considering more advanced techniques.

2 Specializing in Advanced Techniques

Having established the baseline performance of QMix, we now turn our attention to advanced improvements. All experiments in this section were conducted on the `bloxCapture.lay` map.

2.1 Advanced Mixing Networks (QPLEX)

2.1.1 The Challenge: Limitations of Monotonicity

As hypothesized in the reflection of Part 1, the standard QMix architecture enforces a strict monotonicity constraint ($\frac{\partial Q_{tot}}{\partial Q_a} \geq 0$) to ensure the Individual-Global-Max (IGM) principle. While this simplifies the maximization step, it severely limits the function approximation capability of the network. In a Capture the Flag scenario, optimal strategies often require non-monotonic value relationships. QMix cannot represent these value functions, leading to the sub-optimal policies observed previously.

2.1.2 The Solution: QPLEX Implementation

To address this, we implemented **QPLEX** (Duplex Dueling Multi-Agent Q-Learning). QPLEX creates a factorized joint action-value function that satisfies IGM without imposing strict monotonicity on the weights. It achieves this using a duplex dueling architecture, decomposing the joint value Q_{tot} into:

$$Q_{tot}(s, \mathbf{u}) = V_{tot}(s) + A_{tot}(s, \mathbf{u}) \tag{1}$$

where A_{tot} is defined as a weighted sum of individual advantages:

$$A_{tot}(s, \mathbf{u}) = \sum_{i=1}^{n} \lambda_i(s, \mathbf{u}) A_i(s, u_i) \tag{2}$$

Here the weights $\lambda_i(s, \mathbf{u})$ are strictly positive ensuring that if an agent maximizes its individual utility A_i, the global utility Q_{tot} is also maximized. This structure allows QPLEX to maintain the IGM principle while offering a much richer representational capacity than QMix.

We selected QPLEX over alternatives like QTran because QPLEX's transformation approach generally offers better training stability than QTran's soft-constraint relaxation.

Regarding the specific implementation, the architecture processes the global state through a Convolutional Neural Network (CNN) encoder to generate a dense state embedding. This embedding serves as input to the transformation hypernetwork, which generates the mixing weights λ_i. The weights λ_i are enforced to be positive via an absolute value operation on the hypernetwork output. Notably, the hypernetwork is initialized to approximate a Value Decomposition Network (VDN) behavior initially, promoting training stability in the early stages.

2.1.3 Performance Analysis and Reflection

We evaluated the QPLEX agent in two scenarios: first against the `randomTeam` to verify stability and comparison with IQL/QMix, and second against the `baselineTeam` to test performance in a competitive setting. It should be noted that the number of training episodes increased due to the higher complexity.

1. QPLEX vs. Random

QPLEX consistently outperforms the `randomTeam`. While both QPLEX and the improved QMix configuration eventually learn to defeat the random opponent, they exhibit different learning dynamics. QMix converges much faster, reaching a win rate of 0.7 after slightly more than 100 episodes, whereas QPLEX requires around 400 episodes to attain the same winrate. This slower convergence reflects the higher sample complexity induced by QPLEX.

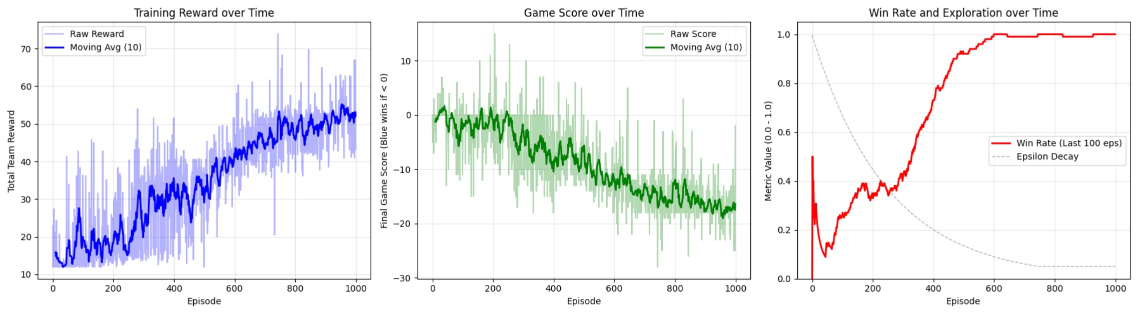


Figure 3: Training results of QPLEX against random opponent on bloxCapture.

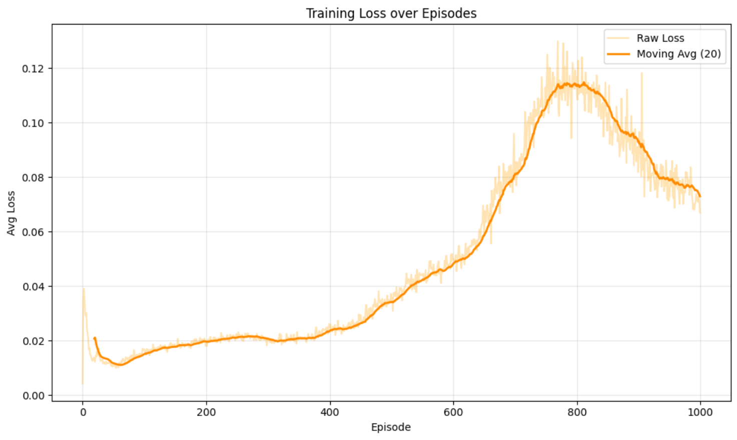


Figure 4: Training Loss evolution for QPLEX architecture. For the sake of conciseness, loss plots are omitted for subsequent experiments. In Value-Based MARL, the magnitude of the TD-error often does not correlate linearly with task performance (as seen here, where loss increases as the agent learns to win), making it a significantly less informative metric compared to win rate or total reward.

3. QPLEX vs. Baseline

Against the much tougher `baselineTeam`, QPLEX manages to reach a win rate of ≈60%. This means that the learned policy has found strategies that can outmaneuver the static rules of the baseline agents.

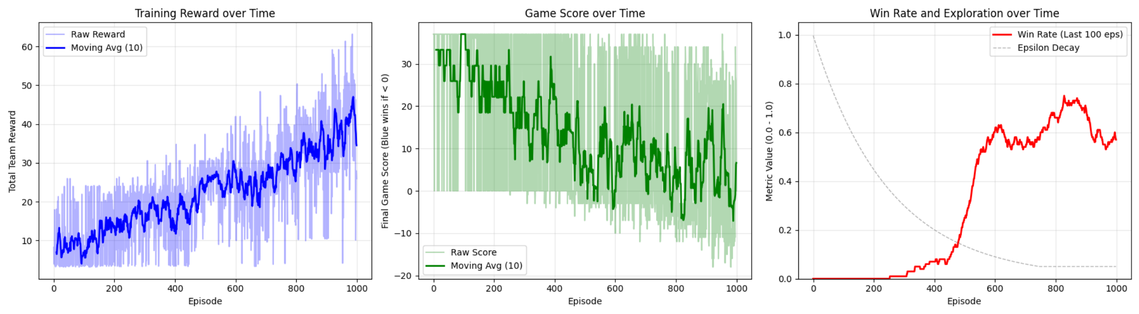


Figure 5: Training results of QPLEX against baseline agent on bloxCapture.

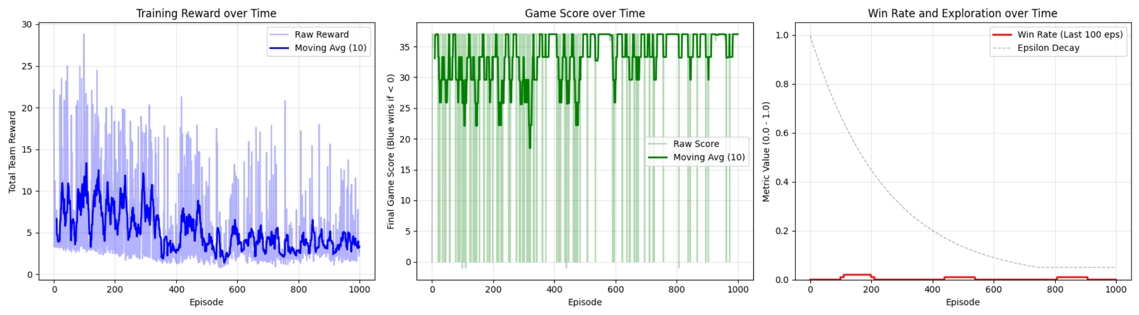


Figure 6: Training results of IQL against baseline agent on bloxCapture.

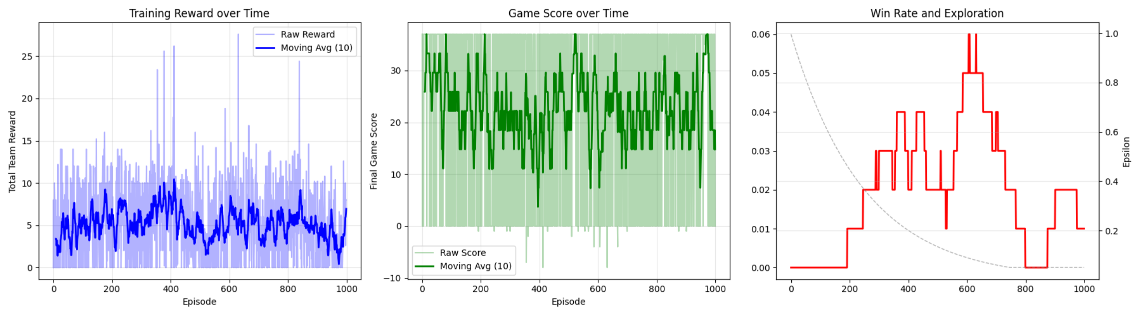


Figure 7: Training results of QMix against baseline agent on bloxCapture.

From a performance perspective, the increased expressivity of QPLEX proves to be decisive in this more challenging setting. While both QMix and IQL fail to learn meaningful strategies against the `baselineTeam`, with win rate never exceeding 0.06 even after extended training, QPLEX is able to consistently beat the baseline. Despite some residual variability, the win rate stabilizes around 0.6–0.7. These results highlight that, although QPLEX requires more training episodes to converge, its richer value decomposition is essential to successfully address strong opponents.

We also analyzed the computational cost by measuring the wall-clock training time normalized per 100,000 steps. IQL, being the simplest architecture with no mixing network, was the fastest at approximately 19 minutes per 100k steps. The centralized training architectures were significantly heavier: QMix required 40 minutes, and QPLEX required 38 minutes (per 100k steps). Theoretically, QPLEX is more complex than QMix as it requires computing $V_i(s)$ and advantages for all actions to determine transformation weights. However, empirically, their training times were comparable, with both being roughly $2\times$ slower than IQL. This suggests that the computational bottleneck in our setup is likely the shared Convolutional Neural Network (CNN) feature extractor rather than the mixing logic itself.

Remark: despite the success shown in Figure 5, we discovered that the standard QPLEX architecture combined with ϵ-greedy exploration is highly sensitive to hyperparameters. To illustrate this fragility, we conducted an experiment using the exact same architecture but slightly modifying a single parameter: the epsilon decay rate, changing it from 0.996 to 0.997.

As shown in Figure 8, this seemingly negligible change resulted in a complete failure to learn, with the win rate flat-lining at 0%. The agents failed to transition effectively from exploration to exploitation, likely getting stuck in local optima before discovering the sparse reward signal.

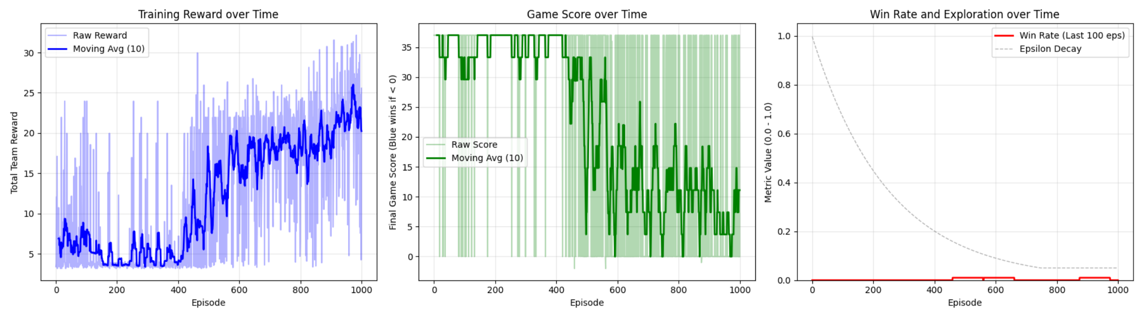


Figure 8: Training failure caused by changing epsilon decay from 0.996 to 0.997. Despite using the advanced QPLEX architecture, the agent fails to learn a winning policy, highlighting the fragility of standard exploration.

This observation suggests that the successful convergence of standard QPLEX relies heavily on ”lucky” random exploration sequences. Lacking a structured exploration incentive, the agent must stumble upon the complex sequence of actions required to capture food purely by chance during the early training phases. If this initial discovery does not happen, the agent inevitably converges to a suboptimal policy during the exploitation phase, reinforcing losing behaviors rather than learning to win. This inherent instability serves as the primary motivation for the next section.

2.2 Coordinated Exploration (Count-Based)

2.2.1 The Challenge: Sparse Rewards and State Space

In the Capture the Flag environment, the primary reward (winning) is sparse and delayed. Standard ϵ-greedy exploration is inefficient in such large state spaces, as agents often fail to encounter the specific sequence of actions required to capture food and return home successfully. As a result, learning frequently collapses into poor local optima. This behavior was repeatedly observed during the early stages of the project, while experimenting with different hyperparameter settings and architectural variants: agents learned to remain in their initial positions to minimize the risk of being captured, thereby avoiding negative outcomes at the cost of never exploring the environment or achieving the task objective.

2.2.2 The Solution: Tabular Count-Based Exploration with Abstraction

To drive exploration, we implemented a **Count-Based Exploration (CBE)** mechanism. This method augments the extrinsic reward from the environment r_{env} with an intrinsic bonus r_{int} inversely proportional to the square root of the visit count $N(s)$ of a state:

$$r_{total} = r_{env} + \frac{\beta}{\sqrt{N(s)}} \tag{3}$$

where β is a tunable scaling factor.

State Abstraction Strategy: Since the raw observation space consists of high-dimensional multi-channel grids, the number of possible states is astronomically large, making a direct tabular approach infeasible. To address this, we implemented a domain-specific state abstraction. Instead of hashing the entire observation tensor, we define the state key s based on three features:

1. **Agent Position:** The (x, y) coordinates of the agent.
2. **Ally Position:** The (x, y) coordinates of the teammate (crucial for coordination).
3. **Enemy Food Context:** To avoid the combinatorial explosion of tracking every food pellet, we abstract the target food map into two boolean sectors: *Top-Half Has Food* and *Bottom-Half Has Food*.

This abstraction allows the agents to recognize "novel situations" (e.g., "I am deep in enemy territory, my ally is defending, and there is food in the top sector") without distinguishing between functionally identical grid patterns. Note that the exploration counts are maintained separately for each agent ID to ensure individual coverage.

2.2.3 Performance Analysis

We conducted experiments comparing the "Simple" state hashing (Agent Position only) against the "Advanced" hashing (Agent Position + Ally Position + Food Context).

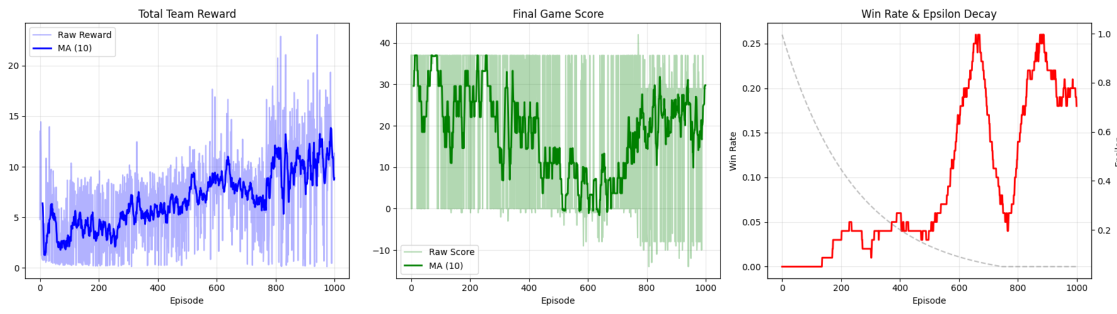


Figure 9: Training results using CBE "simple" against baseline agent on bloxCapture.

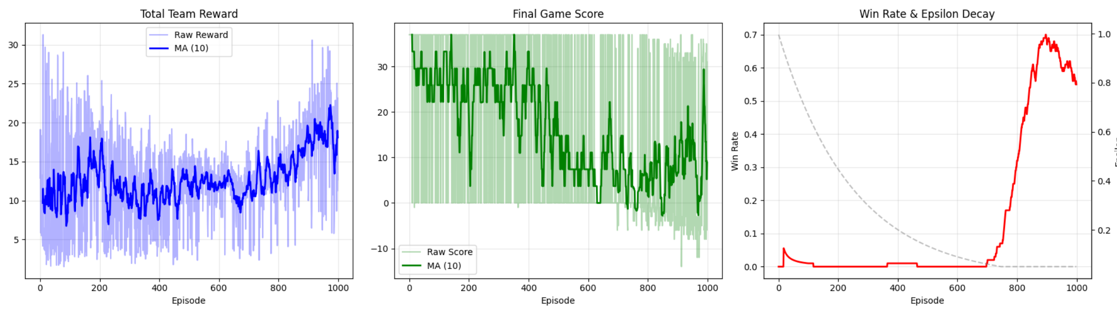


Figure 10: Training results using CBE "advanced" against baseline agent on bloxCapture.

The above images were obtained using an exploration scaling factor $\beta = 0.1$. Using the advanced setting, we then investigated the impact of β by considering a low value ($\beta = 0.01$) and a high value ($\beta = 0.5$). The results are shown below:

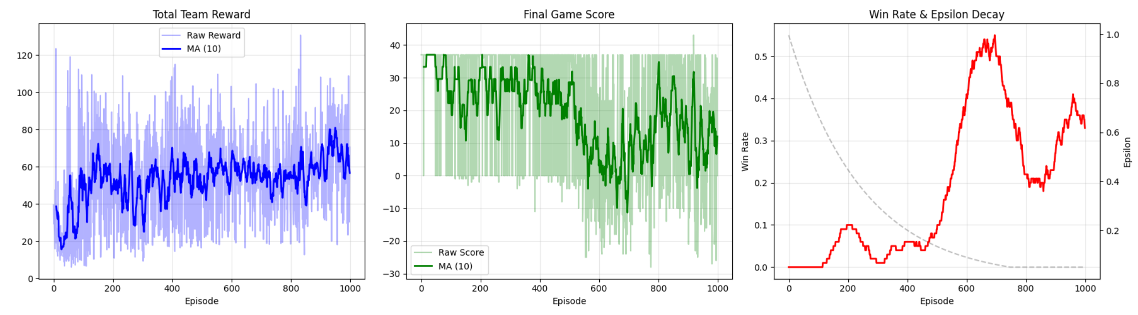


Figure 11: Training results using CBE ”advanced” with $\beta = 0.5$ against baseline agent on bloxCapture.

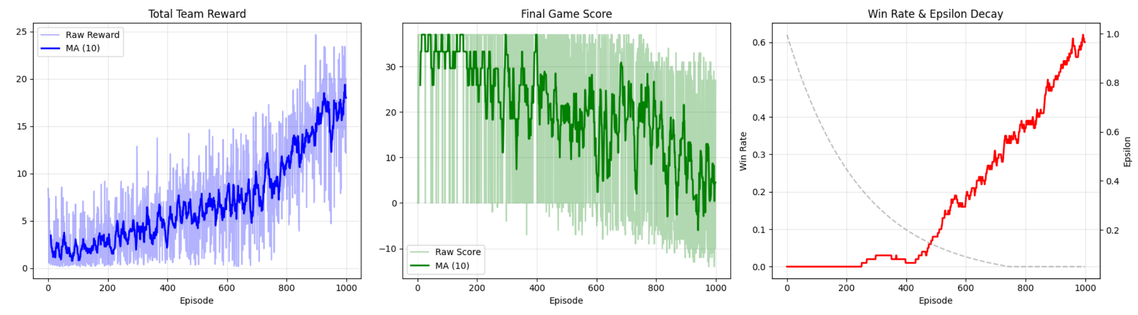


Figure 12: Training results using CBE ”advanced” with $\beta = 0.01$ against baseline agent on bloxCapture.

The results highlight three key insights:

1. **State Representation:** The ”Advanced” hashing significantly outperformed the ”Simple” approach. The Simple hasher suffered from state aliasing: by ignoring ally position and food, it treated different situations as identical. This caused the visit counts $N(s)$ to saturate rapidly, driving the intrinsic exploration bonus to zero prematurely. As a result, the agent experienced catastrophic forgetting: having depended on the exploration bonus to guide its actions, it struggled to switch to exploiting the sparse game rewards, ultimately producing an ineffective policy.

2. **Map Coverage and Strategy Discovery:** The Count-Based Exploration (CBE) bonus led to noticeably better map coverage. Without CBE, agents often remained in the safe “home” area to avoid dying, getting stuck in a local optimum. With CBE active, agents ventured deeper into enemy territory. This behavior was visible during debugging, printing the agents’ position at each step. The increased exploration significantly improved intra-run stability: the agent relied less on random actions to gather food and receive positive rewards, and every run consistently succeeded against the baseline.

3. **Beta Sensitivity:** We observed that high values of β (0.5) destabilized the QPLEX mixer. The large intrinsic bonuses drowned out the sparse reward signal (capturing food), preventing the mixer from learning the true global value function and leading to suboptimal behavior. The agent optimized for ”curiosity” rather than ”winning.” In contrast, a low β (0.01) resulted in steady and stable learning. Even a modest exploration bonus was sufficient to guide the

agent in exploring the map and achieving consistent victories against the baseline. Compared to the initial β (0.1), learning was slower but more reliable.

Consequently, the configuration `QPLEX + CBE (Advanced,` $\beta = 0.01$`)` was selected as the best performing model for this track. While its peak win rate was marginally lower than the best run of the vanilla QPLEX, this configuration offered superior stability and reproducibility. Directly addressing the critical fragility observed in Section 2.1.3, the addition of intrinsic motivation ensured a robust learning curve driven by systematic state coverage, rather than relying on the fortuitous success of random exploration.

2.3 Enhancing Individual Agents

2.3.1 The Challenge: Limitations of Standard DQN

The multi-agent system's performance is fundamentally limited by the stability and sample efficiency of the individual agents. Standard Deep Q-Networks (DQN) often struggle to distinguish between state values and action advantages in complex environments. Furthermore, learning from sparse rewards (such as capturing food) is inefficient when using uniform sampling and standard 1-step returns, leading to slow convergence.

2.3.2 The Solution: Component Integration from Rainbow

To address these issues and maximize the learning potential, we integrated three key components inspired by the Rainbow DQN framework:

- **Dueling DQN:** We split the network's feature extractor into two streams: a Value Stream $V(s)$ and an Advantage Stream $A(s, a)$. This architecture should improve value estimation accuracy.
- **Prioritized Experience Replay (PER):** We replaced the uniform replay buffer with a SumTree-based PER buffer ($\alpha = 0.6$). This prioritizes transitions with high Temporal Difference (TD) error, focusing learning on the most surprising and informative experiences to improve sample efficiency.
- **N-Step Returns:** We implemented N-step returns with $N = 3$. This should help propagate sparse rewards (like eating food) backward much faster than standard 1-step returns.

2.3.3 Performance Analysis

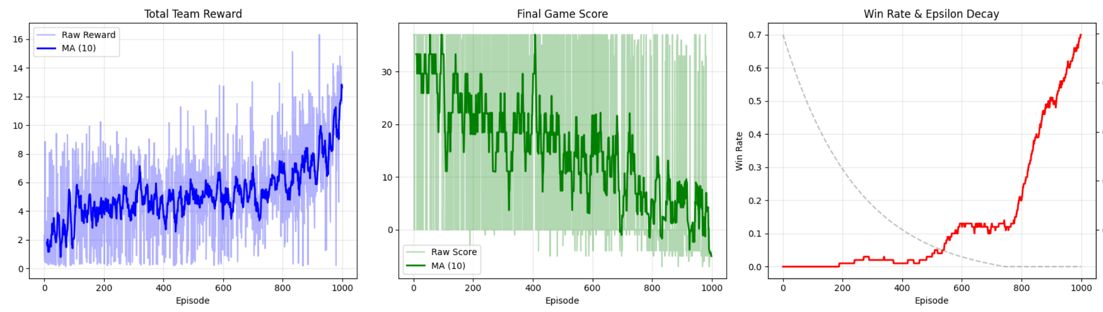


Figure 13: Training results using CBE ”advanced”, Dueling, PER and N-step (against baseline agent on bloxCapture).

Figure 13 shows the training performance obtained by integrating QPLEX and CBE (Advanced) with Dueling DQN, PER, and N-step returns. While the initial warm-up phase was noticeably longer, the full configuration achieved an improvement over the simpler `QPLEX + CBE` setup. After convergence began, the win rate increased rapidly, reaching approximately 0.7, compared to around 0.6 observed with QPLEX + CBE alone.

More importantly, the learning curve shows no clear reduction in slope toward the end of training, suggesting that performance had not yet saturated and could have continued to improve with additional training. This contrasts with the earlier configuration, where learning appeared to stabilize prematurely.

While this combination ultimately improved performance, it required careful hyperparameter tuning. In a multi-agent environment, sources of non-stationarity such as teammates’ exploratory actions can inflate TD errors, making components like PER and N-step returns more sensitive to their respective parameters.

3 Explore and Innovate

Building upon the advanced architecture from Part 2, we identified that while Exploration (CBE) helps find rewards, the sparsity of the feedback signal still makes learning slow and unstable in the early phases. For Section 3, we implemented a comprehensive suite of improvements focused on reward engineering and observation processing, with the aim to accelerate convergence and robustness.

3.1 Potential-Based Reward Shaping (PBRS)

We selected PBRS because it allows us to inject heuristic domain knowledge (e.g., ”carrying food home is good”) into the reward function without altering the optimal policy. Crucially, PBRS maintains policy invariance provided the shaping follows the potential difference form:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{4}$$

where $\Phi(s)$ is a potential function over states. This dense signal guides the agent towards high-value states much faster than waiting for sparse terminal rewards.

We designed a domain-specific potential function $\Phi(s)$ that switches modes based on the agent's context:

- **Hunting Mode (Not carrying food):** $\Phi(s) = -d(agent, food)$. This encourages moving towards the nearest food pellet.
- **Returning Mode (Carrying food):** $\Phi(s) = -d(agent, home) + C + (food\_carried \times bonus)$, where C is a large offset constant. This ensures that picking up food results in a massive jump in potential (triggering a positive reward spike), and carrying it home maximizes that potential.

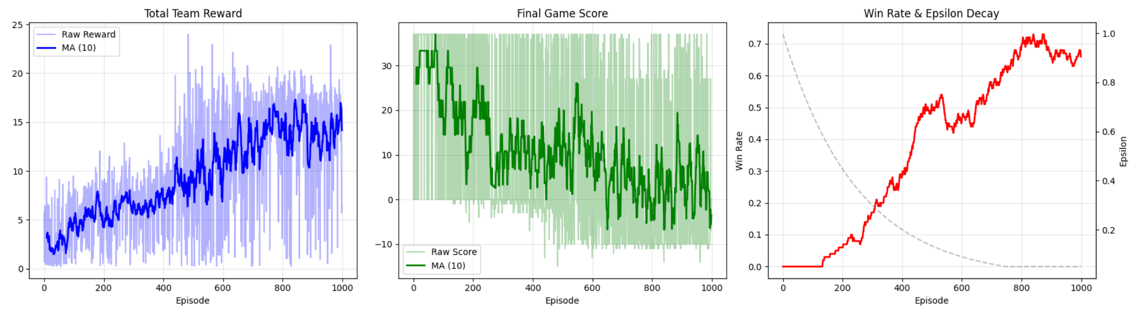


Figure 14: Training results of the fully optimized agent (QPLEX + CBE + Dueling + PER + N-step + PBRS) against the baseline agent.

When comparing this model to the previous best configuration (Figure 13), the integration of PBRS shows two key improvements:

1. **Faster Initial Learning:** The win rate begins its consistent ascent noticeably earlier, before episode 200, confirming that the dense shaping signal effectively guides the agent's policy faster than relying solely on sparse extrinsic rewards.
2. **Higher Peak Performance:** The agent achieves a higher peak win rate of approximately 75% around episode 800, surpassing the ≈70% peak of the previous best model. This suggests PBRS helps the team discover and stabilize a more effective cooperative strategy.

3.2 Supporting Innovations

To complement PBRS, we activated three additional enhancements in our final agent configuration:

1. **Input Normalization:** The observation space is an 8-channel tensor where specific channels encode different game entities (e.g., walls, food, agents). Channel 1, representing the agent's status, uses a hybrid encoding: 0 for empty space, 1 for the agent's presence, and integers > 1 to represent the amount of food carried. This unbounded integer scaling creates a significant magnitude disparity compared to other binary channels. With the idea of stabilizing training, we implemented a custom normalization layer acting solely on this channel, mapping the agent's state to 0.5 + (food carried/20.0). This ensures all inputs remain approximately in the range $[0, 1]$, improving gradient flow.

2. **Time Penalty:** We introduced a small negative reward per step (-0.01). This creates a pressure to finish episodes quickly, discouraging lazy behaviors or deadlocks.

3. **Action Masking:** In the Pacman grid environment, standard agents waste significant training time learning not to walk into walls. To address this, we also implemented explicit Action Masking. The environment provides a list of legal actions $\mathcal{A}_{legal}(s)$ for the current state. During both action selection and training, we force the Q-values of illegal actions to negative infinity:

$$Q_{masked}(s,a) = \begin{cases} Q(s,a) & \text{if } a \in \mathcal{A}_{legal}(s) \\ -\infty & \text{otherwise} \end{cases} \tag{5}$$

   This guarantees that the greedy policy $\pi(s) = \arg\max_a Q_{masked}(s,a)$ always selects a valid move, effectively pruning the search space and stabilizing the learning of the Q_{tot} function.

3.2.1 Performance Analysis

We trained this "Fully Loaded" agent (QPLEX + CBE + Dueling + PER + N-step + PBRS + Normalization + Time Penalty) against the baseline team on the `bloxCapture` map.

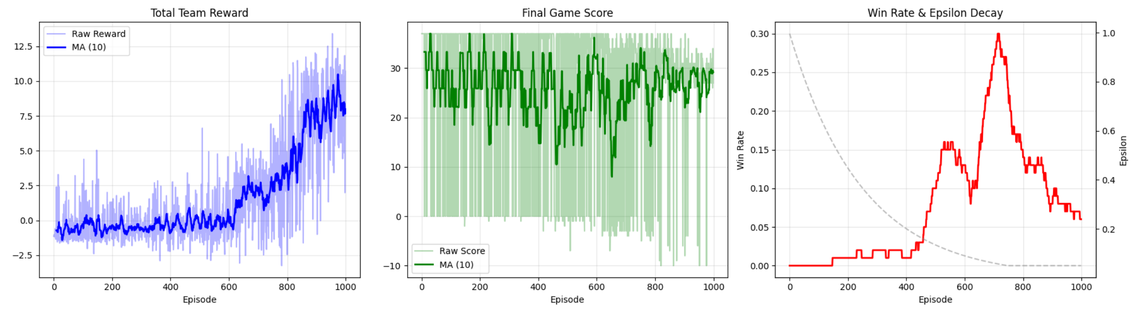


Figure 15: Training results of the fully optimized agent (PBRS, Normalization, Time Penalty) against baseline agent.

The results in Figure 15 demonstrate that, contrary to our expectations, the integration of all advanced components resulted in a regression in performance compared to the best model from Part 2.

- **Low win rate:** The win rate peaks at only ≈ 0.3 (30%), significantly lower than the ≈ 0.7 achieved by the Stage 2 agent.
- **Instability:** After reaching its peak around Episode 800, the performance degrades rather than stabilizing, suggesting catastrophic forgetting or policy oscillation.

3.2.2 Critical Reflection: Why did it fail?

The failure of the fully loaded configuration does not indicate that the introduced components were individually flawed. On the contrary, several of them had already demonstrated clear benefits when evaluated in isolation. The negative outcome instead points to a case of destructive interaction between otherwise effective mechanisms.

1. **PBRS–Normalization Mismatch:** PBRS was shown to be effective when applied in isolation. However, its combination with observation normalization likely altered the geometry of the state space.
2. **Over-constrained Reward Landscape:** The simultaneous presence of dense shaping (PBRS) and a global time penalty imposed multiple, partially competing optimization pressures. While each term individually improves efficiency, their combination proved ineffective.
3. **Complexity Overload:** The QPLEX architecture relies on estimating the joint value function. By adding complex shaping terms, we made the value function significantly harder to approximate. The mixing network likely struggled to decompose this noisy, composite reward signal into meaningful individual credits.

This negative result highlights a critical challenge in Reinforcement Learning: hyperparameter sensitivity and component interference.

In the end, we went back to our best performing architecture (Figure 14) and conducted a generalization test on random layouts. The agent, having been trained exclusively on the fixed `bloxCapture` map, was evaluated over 1000 episodes on unseen random maps without any further training. The experiment yielded a win rate below 20%. While the agents demonstrated basic survival instincts (avoiding collisions with enemies), they failed to navigate effectively or formulate a winning strategy to capture food in the new topologies.
This negative result confirms that our agent has heavily overfitted to the `bloxCapture` layout. Despite using Convolutional Neural Networks (CNNs) which are designed to extract spatial features, training on a single fixed environment caused the network to memorize specific absolute coordinates and trajectories rather than learning generalizable game dynamics (e.g., ”explore unknown areas” or ”pathfind to the nearest dot” regardless of wall configuration).
To solve this in future work, a curriculum learning approach is strictly necessary. The training pipeline should verify the agent on random maps from the very beginning, starting with small, open layouts and progressively increasing complexity. This would force the policy to rely on local observations and relative features (walls vs. food patterns) rather than absolute positions, thus promoting true generalization.

4 Conclusions and Future Work

Through this assignment, we gained deep insights into the challenges of Multi-Agent Reinforcement Learning. We learned that while decentralized execution with centralized training (CTDE) frameworks offer superior theoretical capabilities compared to IQL, their success in practice is contingent upon effective exploration strategies and the careful engineering of the reward signal.

4.1 The Criticality of Hyperparameters

One of the most striking findings across all our experiments was the extreme sensitivity of advanced MARL systems to hyperparameter configurations. Architectural improvements alone proved insufficient without precise tuning of exploration and reward shaping parameters.

- **Exploration Decay:** As discussed in Section 2.1, a seemingly negligible change in the epsilon decay rate (from 0.996 to 0.997) was sufficient to completely prevent learning in QPLEX. This highlights the fragility of MARL methods in sparse-reward settings.

- **Hyperparameter Sensitivity:** In Count-Based Exploration, excessively large values caused the intrinsic reward to dominate the objective, leading to unfocused exploratory behavior. More broadly, we found that tuning the full system was challenging across multiple dimensions: learning rate, replay buffer capacity, target network update rate (τ) all exhibited narrow stability regions. Deviations frequently resulted in divergence, slow learning, or convergence to suboptimal policies.

- **PBRS Parameterization:** PBRS introduced an additional layer of sensitivity. In particular, incorrect scaling or offset selection in the potential function led to several pathological behaviors. Without a sufficiently large offset when switching from hunting to returning mode, the transition associated with picking up food produced a negative shaped reward, causing the agent to avoid collecting food altogether. In other configurations, agents deliberately suicided after acquiring food, as dying reset them to a state with higher potential (closer to home). To mitigate this exploit, we explicitly disabled the PBRS reward when the agent died. Finally, we observed cyclic behaviors where agents maximized shaping rewards by moving indefinitely without delivering food, exploiting the dense potential gradient.

These failures collectively demonstrate that MARL algorithms are not ”magic bullets.” Their theoretical advantages (CTDE, value decomposition) can only be realized when the training schedule, exploration rates, and reward scales are tuned perfectly.

Table 1: Comparative Performance Summary on `bloxCapture` against `baselineTeam`. The table highlights the progression from basic architectures to the peak performance achieved with Rainbow extensions, and the subsequent regression caused by component interference in the ”Fully Loaded” configuration.

Architecture / Configuration	**Win Rate (Peak)**	**Convergence Speed**	**Stability**	**Key Observation**
Part 1: Baselines				
IQL	$< 6\%$	N/A	N/A	Fails to coordinate; greedy local optima.
QMix (Vanilla)	$< 6\%$	N/A	N/A	Representation bottleneck.
Part 2: Advanced Architectures				
QPLEX (Basic)	$\approx 70\%$	Slow ($\approx$ 600 eps)	Modest	Moderate success; high variance.
QPLEX + CBE ($\beta = 0.5$)	$\approx 50\%$	N/A	Low	Intrinsic reward dominance.
QPLEX + CBE ($\beta = 0.01$)	$\approx 60\%$	Still to converge	High	Performance stability.
QPLEX + Rainbow	$\approx 70\%$	Still to converge	High	Good stability and credit assignment.
Part 3: Innovation & Experiments				
QPLEX + Rainbow + PBRS	$\approx 75\%$	Fast	High	Dense reward speeds up initial learning.
All with PBRS (Fully Loaded)	$\approx 30\%$	Fast start, then crash	Low	*Regression:* Signal interference.

4.2 Ideal Methodology

Ideally, had we possessed more computational resources and time, we would have moved away from manual tuning towards a more rigorous optimization strategy:

1. **New algorithmic directions:** we would investigate MAPPO. While our Value-Based approach (QPLEX + PBRS) is powerful, Policy Gradient methods can often handle stochastic

environments and continuous control more smoothly. Additionally, implementing a Curriculum Learning approach where the opponent difficulty scales from Random to Heuristic to Self-Play would likely yield the most robust agents for the final tournament, preventing overfitting to a specific enemy strategy.

2. **Individual Optimization:** Conduct a grid search to find the optimal hyperparameters for each single extension in isolation (e.g., finding the best β for CBE, the best α for PER).
3. **Combinatorial Verification:** Systematically test combinations of extensions to identify destructive interference (as seen in section 3) versus synergistic effects.
4. **Final Tuning & Long Training:** Once the optimal configuration is identified, perform a final hyperparameter sweep on that specific stack and train for a significantly larger number of episodes to reach convergence.

4.3 Plot Twist: The Grand Finale

While the previous section outlined theoretical methodologies for future improvements, we decided to conduct one final, intensive experimental run based on the intuition gained from our negative results with all extensions activated. We hypothesized that the key to ”solving” the environment was to find the perfect tuning.

We manually curated a final configuration designed to prioritize stability and execution speed. By stripping away the components that previously introduced noise or computational overhead, we synthesized an architecture that balances aggressive exploration with strict efficiency constraints.

The ”Apotheosis” Configuration

This final agent was trained with the following specific ”Golden” stack:

- **Architecture:** QPLEX (Duplex Dueling) with Input Normalization.
- **Exploration:** Count-Based Exploration (Advanced) with a high $\beta = 0.1$.
- **Dynamics:** N-Step Returns ($N = 3$) for fast credit assignment, with PER disabled to ensure low-variance updates.
- **Constraint:** A constant Time Penalty (-0.01) to enforce a ”Blitz” strategy.

Final Results

The outcome of this final run exceeded all expectations. As illustrated in Figure 16, the agent achieved a staggering **97% Win Rate** against the `heuristicTeam` on the tournament map (`bloxCapture`).

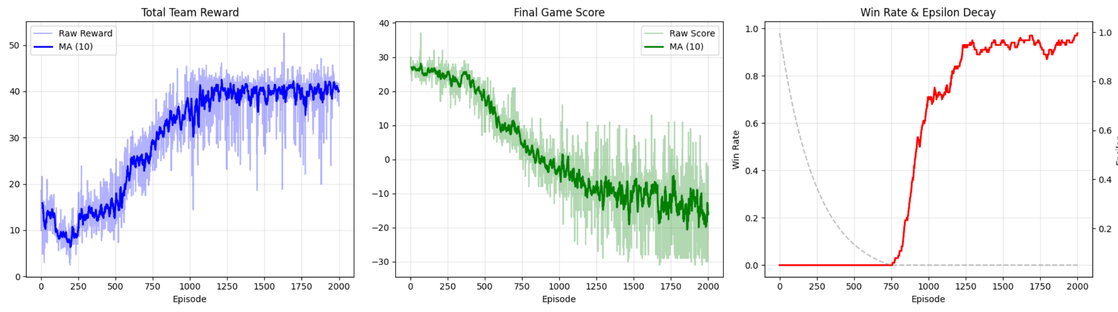


Figure 16: The Apotheosis of our research: By manually tuning the synergy between N-Step Returns and Time Penalty, and removing conflicting shaping signals, the QPLEX agent achieves a near-perfect 97% win rate.

The dominance displayed in this match-up stands as a strong validation of our optimized architecture. It demonstrates that with careful manual tuning and a deep understanding of component interaction, it is possible to achieve high performance even without the computational power of automated grid searches.

References

[1] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. *QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*. Proceedings of the 35th International Conference on Machine Learning (ICML), 2018. Available at: `https://arxiv.org/abs/1803.11485`