Pokemon Battling Agents: Implementation and Discussion of Various RL algorithms

Arzaan Singh
arzaan@tulane.edu

Ian Kreger
ikreger@tulane.edu

Michael Weild
mweild@tulane.edu

Source code repository

Abstract

Competitive Pokémon battles pose a challenging reinforcement learning environment due to partial observability, stochastic transitions, and large action spaces. We implement and compare several RL agents, Tabular Q-Learning, Hierarchical Q-Learning, Linear SARSA, and Deep Q-Networks, using the poke-env interface for Pokémon Showdown. Our experiments show that Tabular Q-Learning significantly outperforms all other implementations, achieving a 98% win rate against the random baseline and over 56% versus the rule-based MaxBasePower agent. Hierarchical and deep agents exhibit strategic promise but suffer from instability and reward exploitation. These results suggest that carefully engineered tabular methods remain highly competitive in constrained Gen 1 battle settings.

1 Introduction

Competitive Pokémon battles pose a sequential decision-making challenge involving uncertainty, opponent modeling, and long-term planning. Each turn, an agent selects an action, either attacking or switching, based on a partially observed game state that includes type matchups, move effects, and potential counterplay. This structure naturally defines the problem as a Markov Decision Process (MDP), making Pokémon an appealing testbed for reinforcement learning (RL) algorithms in adversarial settings.

Pokémon Showdown, along with the `poke-env` Python interface, enables programmatic access to the battle environment and facilitates scalable training of autonomous agents. We design and compare multiple RL-based Pokémon battlers to understand how different learning paradigms influence performance and convergence. We implement Tabular Q-Learning, Linear SARSA, Deep Q-Networks (DQN), and Hierarchical Q-Learning, demonstrating that simpler tabular approaches can outperform more complex methods in the constrained dynamics of Generation 1 gameplay.

2 Related work

Competitive Pokémon battling has recently transitioned from a community hobby to a structured research domain for evaluating reinforcement learning agents. The PokéAGENT competition formalized this setting by providing standardized battle environments, leaderboards, and reproducible evaluation protocols [Karten et al., 2025]. Early academic efforts, such as *Optimal Battle Strategy in Pokémon using Reinforcement Learning*, demonstrated that Generation 1 battles could be modeled as a Markov Decision Process and explored Q-learning for move selection and tactical planning [Kalose et al., 2018]. Metamon extended this paradigm by introducing scalable adversarial self-play, enabling agents to train over thousands of encounters and reveal emergent counter-strategies [Grigsby, 2024], although these systems remained far from consistently surpassing strong human players.

Preprint. Under review.

More sophisticated pipelines have since been proposed. An MIT thesis reframed battle optimization using a hybrid Monte Carlo Tree Search and RL architecture to support long-horizon reasoning and opponent modeling [Wang, 2024]. These works collectively highlight Pokémon battles as a challenging RL benchmark due to high branching factors, delayed rewards, and partial observability.

Frameworks such as `poke-env` have facilitated reproducible experimentation by offering state extraction, action encoding, and Pokémon Showdown connectivity. However, these libraries serve primarily as infrastructure rather than sources of advanced agents. Recent efforts like PokeChamp explore Bayesian priors, large language models, and hybrid optimization to improve generalization across opponents and formats [Kannan et al., 2025], though performance remains unstable and lacks consensus on best practices.

3 Problem formulation

We model competitive Pokémon battles as a finite-horizon Markov Decision Process (MDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the set of legal actions, $\mathcal{P}$ the transition dynamics induced by the Pokémon Showdown simulator, $\mathcal{R}$ the reward function, and γ the discount factor. A battle terminates when all Pokémon on one team faint, yielding a binary outcome, or when a player exceeds the 10-second turn timer. The environment is adversarial, partially observable, and combinatorial, requiring the agent to reason about type matchups, switching, move accuracy, status effects, and long-term board control under stochastic Generation 1 mechanics.

3.1 State space

Each state s_t is a feature vector extracted from `poke-env`. Depending on the implementation, features include the identity of active Pokémon, remaining HP (bucketed), stat boosts, type relationships, status conditions (e.g., `SLP, FRZ`), and move metadata such as accuracy and base power.

In tabular Q-learning, we represent states using discrete encodings of key battle attributes (e.g., Pokémon ID, move ID). This yields a 6-dimensional state vector:

$$s_t = \langle \texttt{my\_hp}, \texttt{opp\_hp}, \texttt{my\_type}, \texttt{opp\_type}, \texttt{opp\_status}, \texttt{im\_faster} \rangle.$$

Our hierarchical Q-learning agent decomposes the decision process into a master agent that selects among five macro-actions (four moves and a switch) and a subagent that selects which Pokémon to switch into. Their state representations are:

$$s_t^{\text{master}} = \langle \text{self\_mon}, \text{opp\_mon}, \text{self\_hp\_bucket}, \text{opp\_hp\_bucket} \rangle,$$
$$s_t^{\text{sub}} = \langle \text{party\_pokemon}, \text{opp\_pokemon} \rangle. \quad (62)$$

For function approximation methods (DQN and Linear SARSA), we replaced discrete encodings with continuous features such as normalized HP, one-hot move types, damage estimates, and type-effectiveness scores. Feature dimensionality ranged from 21 to over 600, with hidden layers of 128–512 units in DQN variants. While Generation 1 constraints keep these representations tractable, the resulting state spaces are still large enough to prevent full convergence.

3.2 Action space

At each turn, the agent selects $a_t \in \mathcal{A}(s_t)$, corresponding to one of the available moves (up to four) or a switch into any unfainted party member. Because available actions depend on team state and game progression, $\mathcal{A}(s_t)$ is non-stationary, a property that makes the domain more challenging than fixed-action benchmarks such as Atari. This also explains why hand-engineered bots like `maxbp` perform competitively in Generation 1 but fail to generalize beyond their encoded heuristics.

3.3 Reward structure

We initially employed a sparse terminal reward:

$$R(s_t, a_t) = \begin{cases} +1, & \text{win}, \\ -1, & \text{loss}. \end{cases}$$

However, the absence of intermediate feedback resulted in slow learning and brittle policy discovery. To accelerate credit assignment, we introduced reward shaping:

$$R(s_t, a_t) = \begin{cases} +100, & \text{win}, \\ -100, & \text{loss}, \\ +20, & \text{opponent fainted}, \\ -20, & \text{self fainted}, \\ 0.5 \cdot \Delta\text{HP}, & \text{damage dealt}, \\ -0.5 \cdot \Delta\text{HP}, & \text{damage taken}. \end{cases}$$

While effective, this formulation introduced reward-hacking behaviors such as repeated healing or status moves that accrued points without improving board position. To mitigate this, we added auxiliary penalties and bonuses:

$$R'(s_t, a_t) = \begin{cases} -30, & \text{repeat move more than three times}, \\ +10, & \text{inflict a new status}. \end{cases}$$

3.4 Objective

The agent seeks a policy $\pi(a \mid s)$ maximizing the expected discounted return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \right],$$

where T is the battle horizon. All Q-learning and SARSA variants were trained using online temporal-difference bootstrapping, while DQN replaces tabular updates with a neural value approximator trained from replayed transitions. Since $\mathcal{P}$ is unknown and opponent behavior is stochastic, this constitutes a model-free, adversarial RL setting requiring policy learning under uncertainty.

4 Preliminary solutions

We implemented multiple reinforcement learning agents, progressing from tabular methods to hierarchical extensions, neural approximators, and an LLM-augmented approach. Each model reflects a different hypothesis about how strategic knowledge in Pokémon battles should be represented and propagated.

4.1 Tabular Q-learning

Our baseline tabular Q-learning agent maintains a value table $Q(s, a)$ updated via

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right].$$

All variants used linear ϵ-decay from 1.0 to 0.05 and ϵ-greedy action selection. The initial implementation, restricted to a win/loss reward and no switching, served primarily as an environment sanity check and achieved negligible performance.

Introducing intermediate rewards led to modest improvement (=36% vs. random), but also revealed reward hacking behaviors (e.g., repeated healing/status moves). Our final version incorporated switching and additional penalties to suppress these exploits. Despite enlarging the state space significantly, this model achieved our best results: approximately 98% win rate vs. `random` (Figure 6) and 55% vs. `maxbp` (Figure 5). Additional training and refined shaping could plausibly yield further gains.

4.2 Hierarchical Q-learning

To mitigate the combinatorial action space, we decomposed decisions into a master policy (selecting among four moves and "switch") and a subpolicy that chooses the replacement Pokémon. This division drastically reduces the effective Q-table size and stabilizes early learning. As shown in

Figure 1, the hierarchical agent quickly surpassed 80% vs. `random`, but required millions of battles to improve further due to persistent exploration demands and slow convergence. Earlier variants with larger state encodings (Figures 2–3) failed to converge even after 2M battles, demonstrating sensitivity to representation choices.

4.3 SARSA and Deep Q-network (DQN)

Our Linear SARSA and DQN models approximate $Q(s, a)$ in feature space rather than enumerating discrete states. Both used replay buffers, while DQN additionally employed a target network for stabilization. Linear SARSA plateaued at =20% against `random` (Figure 9) due to its inability to learn switching behaviors. DQN, however, captured nonlinear patterns and switching strategies, reaching =85% win rate vs. `random` (Figure 8). Nonetheless, DQN was highly sensitive to exploration schedules and prone to policy collapse when replay distributions became skewed.

4.4 LLM-based reasoning

We attempted to integrate the PokéChamp LLM-based opponent-modeling framework, which conditions action choices on predicted movepools. However, the system was incompatible with Generation 1 mechanics, and dynamic behaviors such as `Ditto` transformations repeatedly crashed the opponent-model module. Moreover, running the LLM introduced prohibitive latency (=4 minutes per game) and incurred significant token costs, especially in battles that terminated via the server's 1000-action limit. These constraints prevented systematic evaluation.

5 Evaluation

We evaluated all agents in the Pokémon Showdown Gen 1 environment using the `poke-env` API. Each model was trained and tested across multiple random seeds, and results were logged to CSV files containing win rates, exploration values, and auxiliary diagnostics. Performance was measured against two standard opponents: the `random` baseline and the rule-based `maxbp` agent.

5.1 Tabular Q-Learning

Tabular Q-Learning exhibited the strongest performance among all models. Early variants without switching and with sparse rewards plateaued around 36% win rate. Introducing shaped rewards and enabling switching substantially improved strategic depth and led to consistent policy improvement. The final trained agent reached approximately 98% win rate versus `random` and 55–56% versus `maxbp` (Figures 6, 5). These results are notable given that battles involve randomly sampled teams, making some scenarios unwinnable by construction.

Despite these gains, the agent remained sensitive to reward design: penalties applied upon switching discouraged long-term advantageous swaps, occasionally biasing the policy toward staying in with unfavorable matchups. A refined reward structure that distinguishes tactical damage from positional disadvantage would likely yield additional improvements.

5.2 Hierarchical agent behaviors

The hierarchical Q-learning agent quickly achieved strong performance against the `random` opponent, initially exceeding 80% win rate. However, its learning dynamics were unstable: sparse rewards and insufficient exploration caused Q-values to propagate poorly, producing intermittent policy collapse during later training (Figure 1). Variants with larger state encodings failed to converge even after millions of battles, suggesting that while decomposing the action space reduces representational burden, careful reward shaping and slower decay schedules are necessary for stable convergence.

5.3 Linear SARSA and DQN results

Linear SARSA improved sample efficiency relative to tabular methods, but consistent reward hacking behaviors led to premature convergence toward degenerate policies, often involving repeated moves that maximized intermediate rewards without increasing win rate. Most variants eventually collapsed.

DQN was more expressive and successfully learned switching behaviors, reaching win rates of approximately 85% against `random` (Figure 8). However, performance proved highly sensitive to feature design and replay buffer composition. When the distribution of stored transitions became homogeneous, the policy collapsed, illustrating that deep function approximation in this domain requires significantly more tuning than tabular methods. Gen 1 Pokémon is deceptively simple but structurally hostile to deep RL without dense reward shaping.

Table 1: Performance of implemented agents against baseline opponents in Pokémon Showdown (Gen 1).

Agent	vs Random	vs MaxBP	Behavior
Tabular Q-Learning	**98%**	**55–56%**	Stable, improving
Hierarchical Q-Learning	80–90%	$<$20%	Partial convergence
Deep Q-Network (DQN)	$\sim$85%	$\sim$25%	Sensitive, collapses
Linear SARSA	$\sim$20%	$<$5%	No convergence

6 Conclusion

Our results show that, within the constrained mechanics of Generation 1 Pokémon, tabular reinforcement learning remains remarkably competitive. Tabular Q-Learning achieved the best performance, reaching a 98% win rate against the `random` agent and 56% against `maxbp`, outperforming hierarchical, linear, and deep variants. While hierarchical methods reduced action-space complexity, they exhibited unstable convergence, and both DQN and Linear SARSA suffered from reward hacking and policy collapse. Future work should explore more expressive reward shaping, improved exploration strategies, and scalable state abstractions to support larger team spaces and later Pokémon generations.

Workload division

- **Arzaan Singh**: Implemented the Hierarchical Q-Learning, Linear SARSA, and DQN.
- **Ian Kreger**: Ian setup and handled environment integration with `poke-env`, and developed the Tabular Q-Learning.
- **Michael Weild**: Studied the LLM-assisted action reasoning and helped develop the DQN implementation, running and debugging the model.
- **All authors**: Participated in experimentation, result interpretation, and the report.

References

Luca Castronovo. poke-env documentation. https://poke-env.readthedocs.io/en/stable/getting_started.html#connecting-bots-to-showdown, 2024. Accessed: 2025-12-06.

Matteo Dell'Acqua. Alphapoke project delivery. https://matteoh2o1999.github.io/en-us/projects/alphaPoke-project-delivery, 2025. Accessed: 2025-12-06.

Jake Grigsby. Metamon: Reinforcement learning framework for pokémon. https://metamon.tech/, 2024. Accessed: 2025-12-06.

Akshay Kalose, Kris Kaya, and Alvin Kim. Optimal battle strategy in pokémon using reinforcement learning. Technical report, Stanford University, 2018. AA228 final project report, accessed 2025-12-06.

Sarthak Kannan, Zhibo Chen, Hanxiao Hu, Yiming Chen, Zihan Dai, Haotian Sun, Xiaohui Zhai, and Yuanzhi Wang. PokéChamp: an expert-level minimax language agent for competitive pokémon, 2025. URL arxiv.org.

Seth Karten, Jake Grigsby, Stephanie Milani, Kiran Vodrahalli, Amy Zhang, Fei Fang, Yuke Zhu, and Chi Jin. The pokeagent challenge: Competitive and long-context learning at scale. In *NeurIPS Competition Track*, April 2025.

Guangcong Zarel Luo. Pokémon showdown. `https://pokemonshowdown.com/`. Accessed: 2025-12-06.

Brandon Tan and Dhruv Bhatt. Reinforcement learning for pokémon red. Technical report, Stanford University, 2018. AA228 Final Project Report, Accessed: 2025-12-06.

Author(s) Unknown. Scalable offline reinforcement learning for pokémon battles. *arXiv preprint arXiv:2503.04094*, 2025. URL `https://arxiv.org/abs/2503.04094`. Accessed: 2025-12-06.

Jett Meng Wang. Winning at pokemon battles: Reinforcement learning with monte carlo tree search and proximal policy optimization. Master's thesis, Massachusetts Institute of Technology, 2024. Accessed: 2025-12-06.

A Appendix

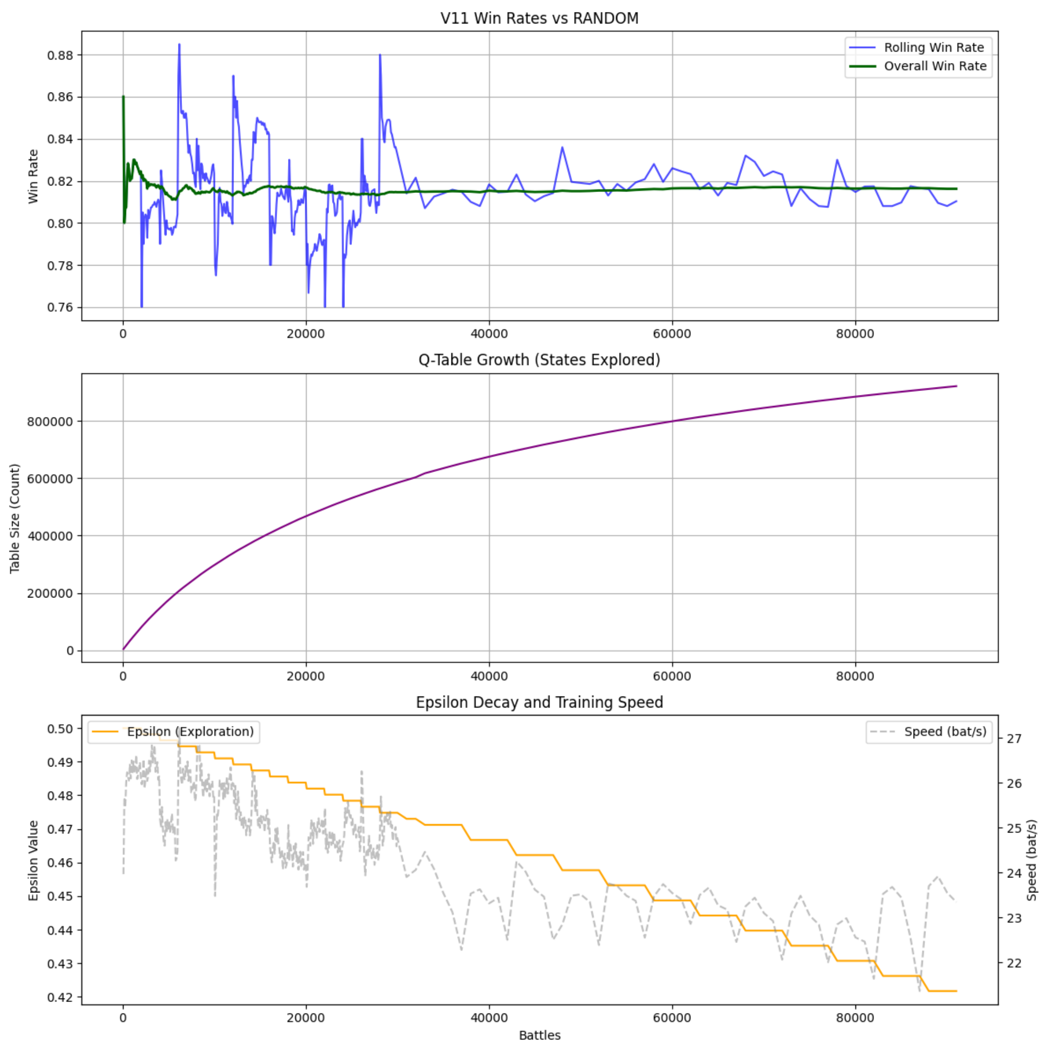

Figure 1: Hierarchical Q-learning training metrics: rolling and overall win rates (top), Q-table state growth (middle), and epsilon decay with training speed (bottom).

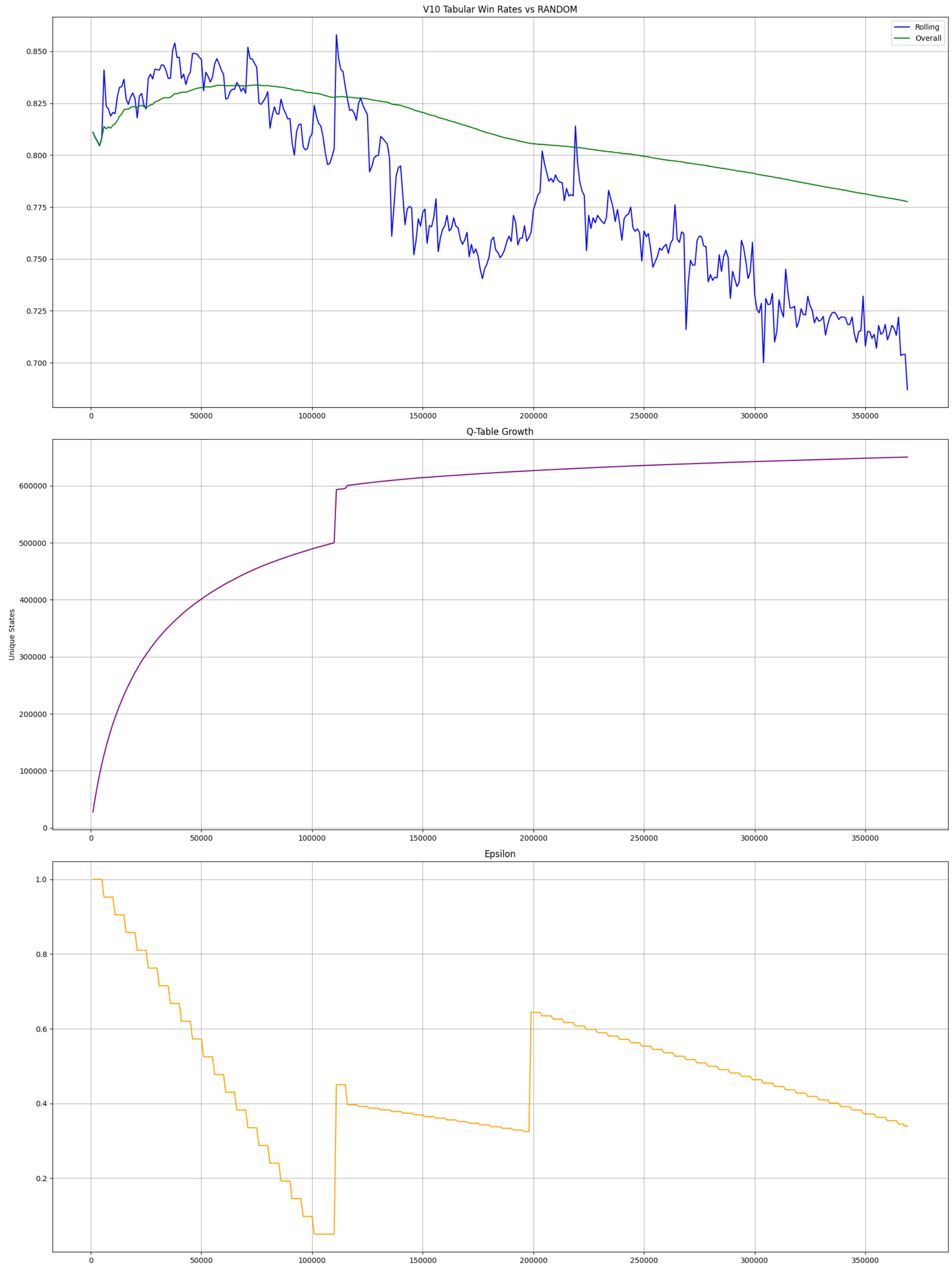

Figure 2: Hierarchical Q-learning training metrics for First Failed Version Against Random: rolling and overall win rates (top), Q-table state growth (middle), and epsilon decay with training speed (bottom).

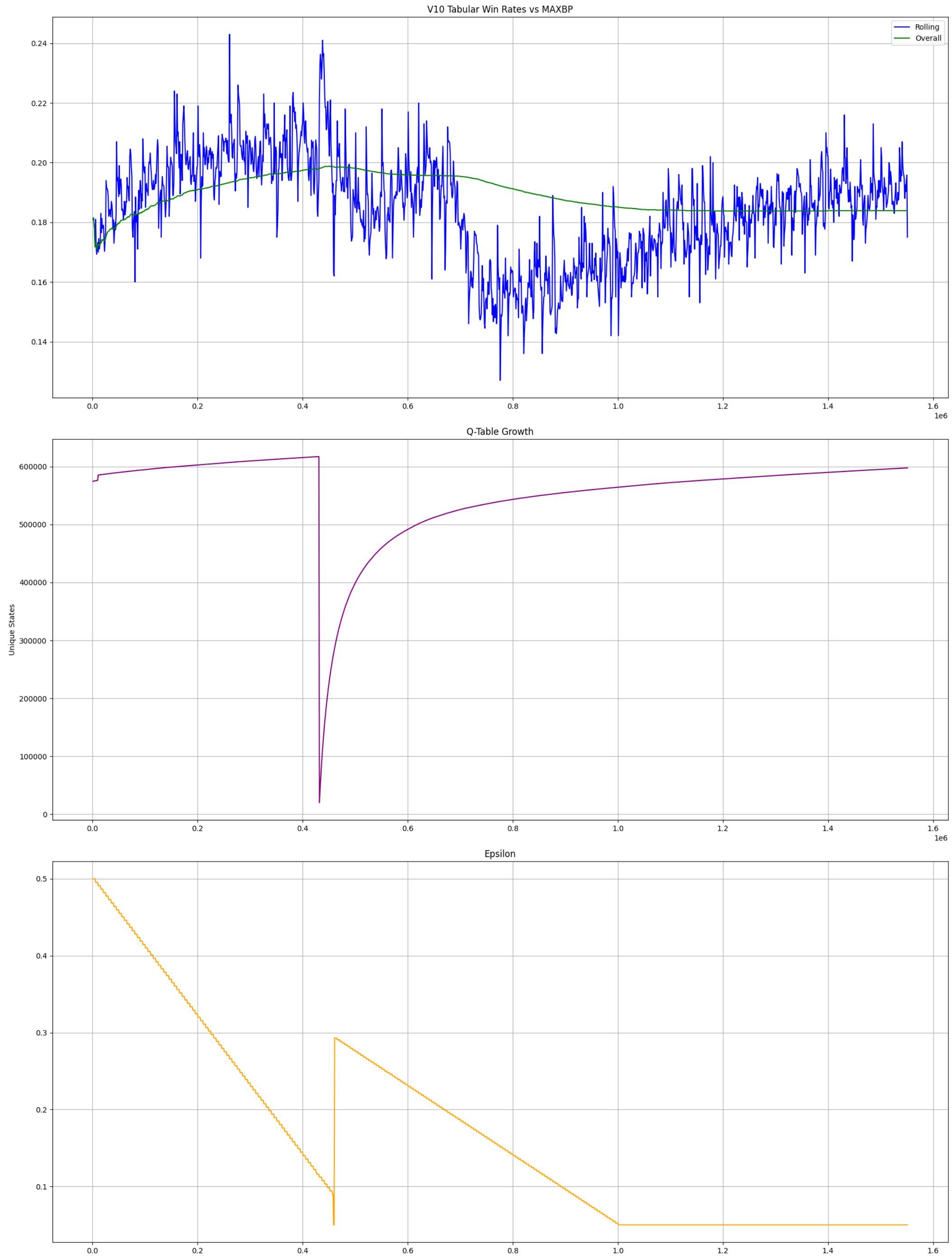

Figure 3: Hierarchical Q-learning training metrics for First Failed Version Against MaxBP: rolling and overall win rates (top), Q-table state growth (middle), and epsilon decay with training speed (bottom).

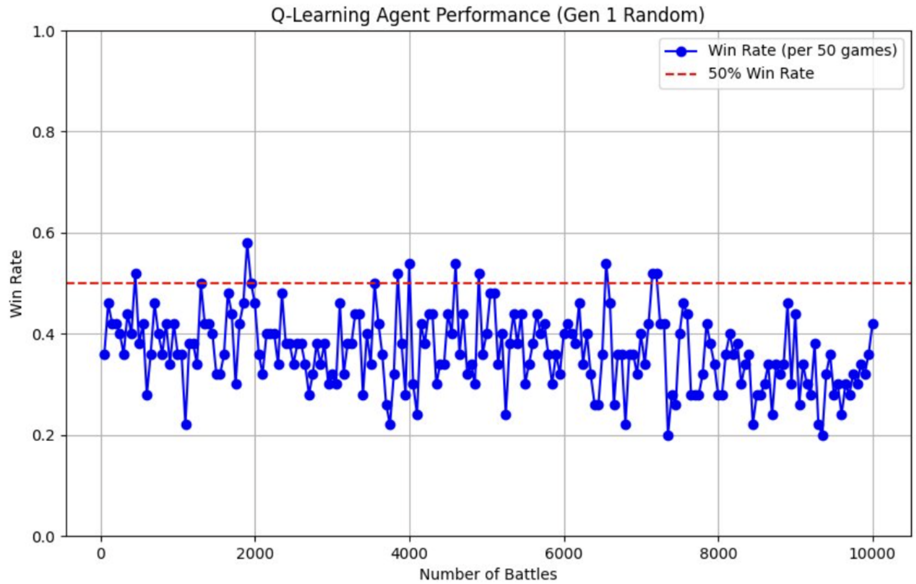

Figure 4: Tabular Q-learning training metrics: Implementation 2 Rolling Win Rate vs Random Player

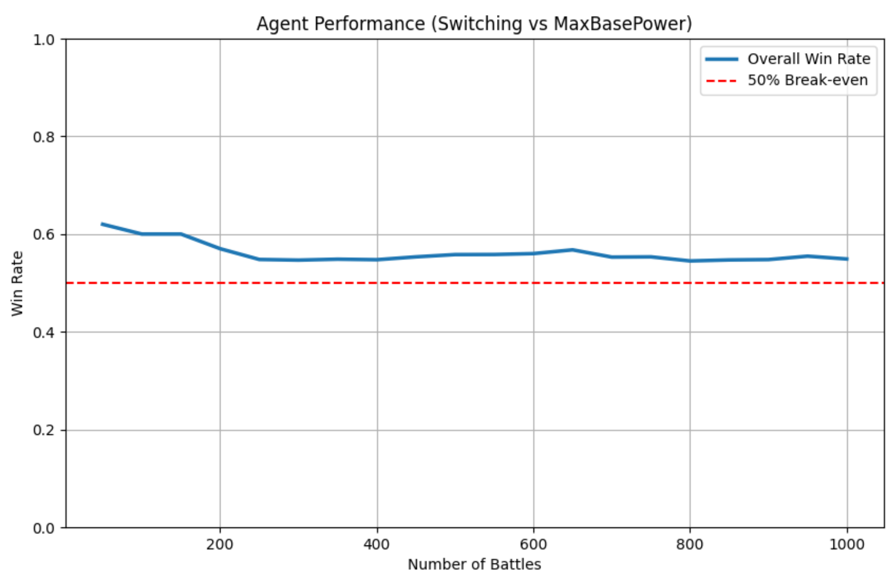

Figure 5: Tabular Q-learning training metrics: Implementation 3 Overall Win Rate vs Maximum-BasePower Player

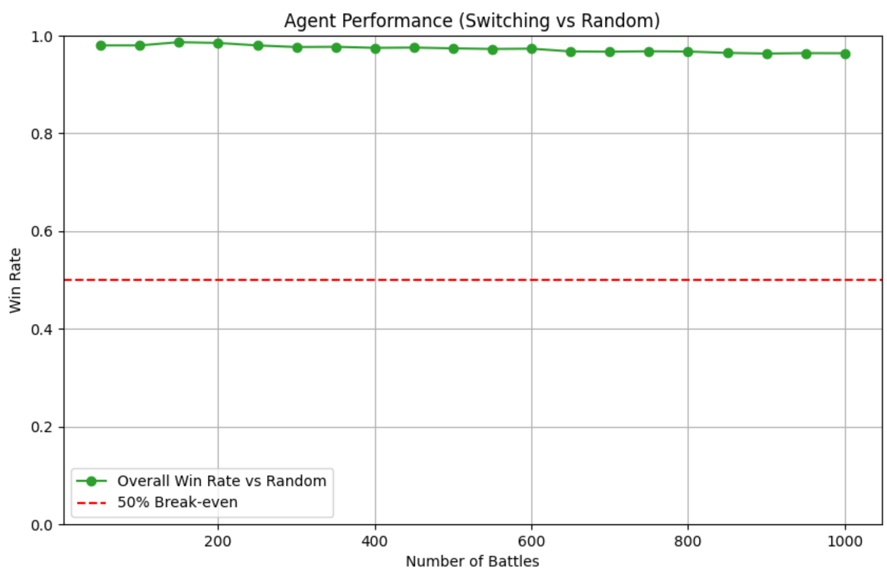

Figure 6: Tabular Q-learning training metrics: Implementation 3 Overall Win Rate vs Random Player

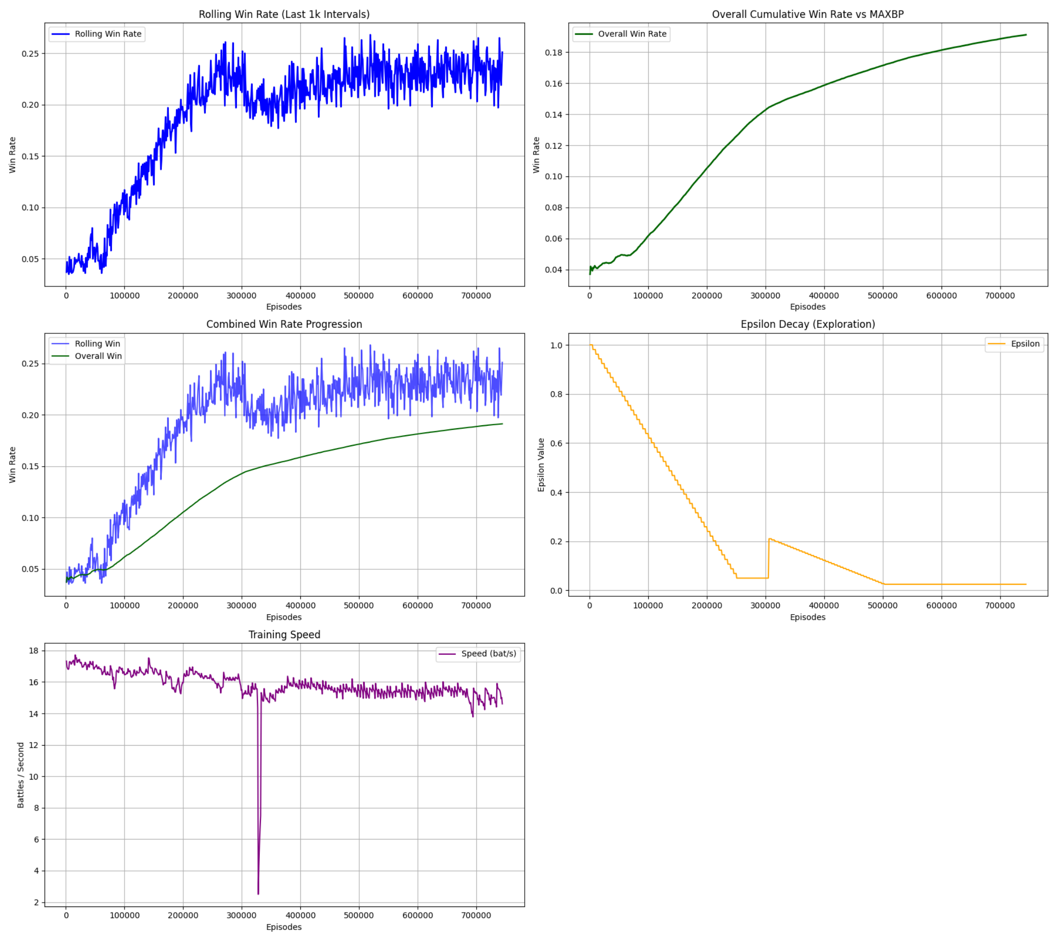

Figure 7: DQN training metrics against MaxBasePower: rolling win rate (top left), overall win rate progression (top right), combined win rate comparison (middle), epsilon decay (right middle), and training speed in battles per second (bottom).

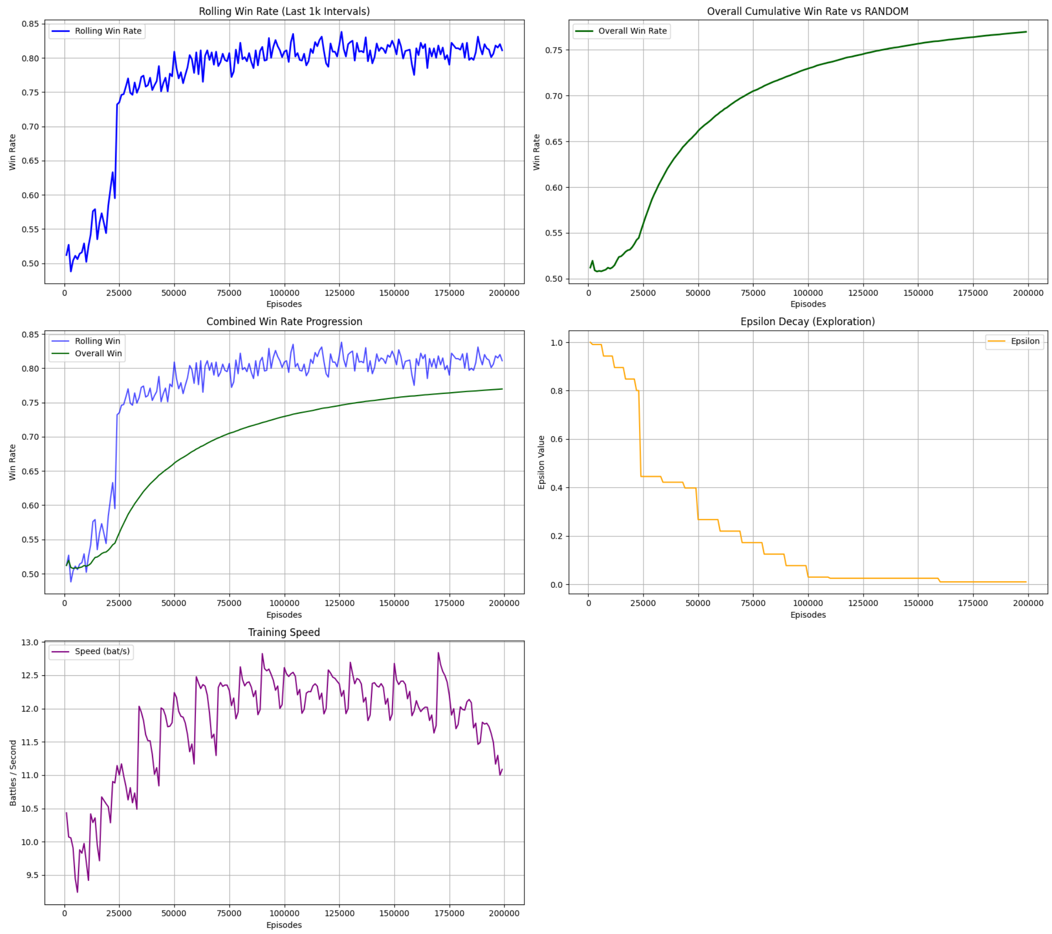

Figure 8: DQN training metrics against Random agent: rolling win rate (top left), overall win progression (top right), combined win analysis (middle), epsilon decay (right middle), and training throughput (bottom).

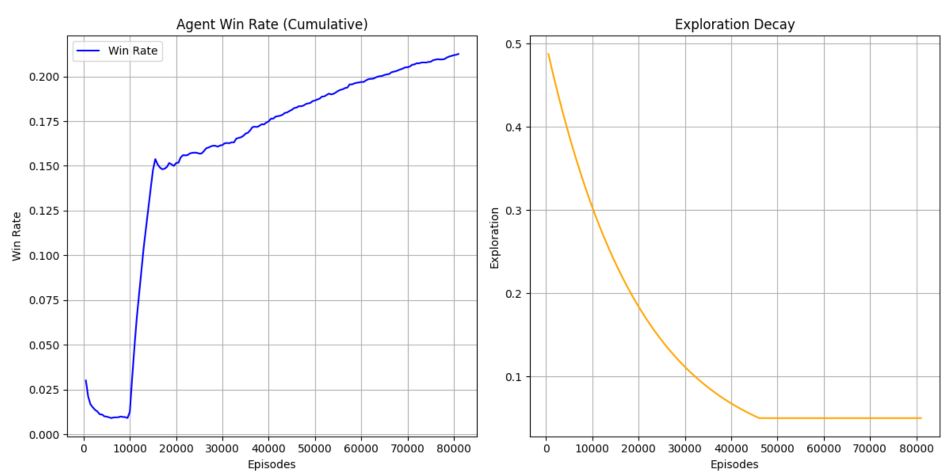

Figure 9: SARSA agent performance against the MaxBP. Left: cumulative win rate showing slow initial convergence followed by consistent improvement once exploration decreases. Right: exponential epsilon decay schedule demonstrating diminishing exploration over training episodes.