Evaluating the Impact of Prioritized Experience Replay on Deep Q-Learning Performance in Tetris

Neil Adrian Baltar
College of Computing and Information Technology (CCIT) National University (N.U.)
Manila, Philippines
baltarnb@students.national-u.edu.ph

Carl Arvin Hipolito
College of Computing and Information Technology (CCIT) National University (N.U.)
Manila, Philippines
hipolitocc@students.national-u.edu.ph

Xavier Pagdanganan
College of Computing and Information Technology (CCIT) National University (N.U.)
Manila, Philippines
pagdangananxp@students.national-u.edu.ph

***Abstract*—This study presents the development of an enhanced Computer Laboratory Monitoring System integrated with machine learning for predictive maintenance. The proposed system aims to assist laboratory administrators by automatically predicting the condition of computer components based on historical and operational reports, without relying on image data. A Deep Q-Network (DQN) was implemented as the learning model, utilizing two distinct training approaches: Uniform Experience Replay (UER) and Prioritized Experience Replay (PER). UER treats all experiences equally, while PER prioritizes significant experiences with higher temporal-difference errors to accelerate convergence. Both approaches were trained under identical parameters to analyze their impact on training efficiency and model accuracy. Experimental results demonstrated that PER provided more stable learning performance and faster reward optimization compared to UER. The integration of predictive intelligence into the monitoring system improves maintenance scheduling, reduces downtime, and contributes to more efficient management of computer laboratory resources.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for developing intelligent agents capable of learning optimal behaviors through interaction with dynamic environments [1]. Among various RL approaches, the Deep Q-Network (DQN) algorithm has gained prominence for its ability to approximate complex value functions using deep neural networks, enabling effective decision-making in high-dimensional state spaces [2]. The success of DQN in domains such as Atari 2600 games [3] and robotic control has inspired numerous studies aimed at improving its learning efficiency, stability, and convergence rate.

A critical component in the DQN framework is the *experience replay* mechanism, which stores past transitions (s, a, r, s') in a replay buffer for randomized sampling during training [4]. This approach mitigates temporal correlations in sequential data, enhances sample efficiency, and stabilizes learning. However, traditional uniform sampling in replay memory treats all experiences equally, which may limit learning effectiveness, especially when some transitions provide more significant learning signals than others [5]. To address this limitation, Prioritized Experience Replay (PER) was introduced to bias the sampling probability toward transitions with higher temporal-difference (TD) errors [6]. By focusing on more informative experiences, PER aims to accelerate learning and improve the overall policy performance.

Tetris serves as an ideal testbed for reinforcement learning research due to its discrete state space, complex long-term dependencies, and stochastic dynamics [7]. The game requires strategic decision-making to optimize line clearances and prevent premature termination, making it a challenging benchmark for evaluating learning-based algorithms. Although several RL studies have explored Tetris using algorithms such as Q-learning, Sarsa, and policy-gradient methods, the comparative analysis of experience replay strategies within a DQN framework remains relatively underexplored.

This study aims to investigate the performance differences between *standard replay memory* and *prioritized experience replay* in training DQN agents to play Tetris. Both replay strategies are implemented under identical network architectures and hyperparameters to ensure a fair comparison. The agents are trained in the Gymnasium-based Tetris environment, with performance evaluated using metrics such as average episode reward, number of cleared lines, and training stability. The results of this comparative analysis are expected to provide insights into how replay memory design influences the efficiency and effectiveness of deep reinforcement learning agents.

The remainder of this paper is organized as follows: Section II discusses related literature on reinforcement learning and experience replay techniques. Section III describes the methodology, including environment configuration, DQN architecture, and replay memory implementations. Section IV presents the experimental setup and results. Finally, Section V concludes the paper and outlines potential directions for future work.

II. Review of Related Literature

A. Reinforcement Learning (RL)

Reinforcement Learning (RL) is a subfield of machine learning in which an agent learns to make sequential decisions by interacting with an environment and receiving scalar feedback signals known as rewards. The primary objective of the agent is to maximize cumulative rewards over time by learning an

optimal policy that maps states to actions [1]. RL problems are typically modeled as Markov Decision Processes (MDPs), which define the set of states, actions, transition dynamics, and reward functions governing the agent–environment interaction [8].

Unlike supervised learning, where labeled data guide the model, RL relies on experience-based exploration and feedback to improve performance. This makes RL particularly suited for dynamic and uncertain environments where explicit labels are unavailable [9]. However, RL also faces several challenges, including the trade-off between exploration and exploitation, sparse or delayed rewards, and high-dimensional state spaces [10]. These challenges are especially apparent in game-based environments, where agents must plan actions over long horizons to achieve success.

The theoretical foundations of RL have been extensively discussed by Sutton and Barto [1], who emphasized the importance of temporal difference learning, reward prediction, and policy optimization. These principles continue to guide modern RL applications in domains such as robotics, game playing, and autonomous control systems.

B. Game Environments for Reinforcement Learning

In reinforcement learning (RL) research, game environments play a crucial role as testbeds for developing, benchmarking, and evaluating algorithms. These environments provide controlled yet complex scenarios that allow agents to learn optimal policies through repeated interactions. The design of the environment—including its reward structure, state representation, and action space—significantly influences the learning performance and generalization capabilities of RL models [11]. Classic game environments such as Atari 2600, OpenAI Gym, and MuJoCo have served as standard platforms for testing reinforcement learning algorithms, fostering reproducibility and consistent performance evaluation across studies [12].

1) OpenAI Gym and Gymnasium: OpenAI Gym, introduced by Brockman et al. [13], is one of the most widely adopted environments for reinforcement learning research. It provides a unified interface for a diverse set of tasks, ranging from simple control problems like CartPole to complex Atari games. The modular structure of Gym enables researchers to focus on algorithm development without worrying about environment implementation details. In 2022, Gymnasium was introduced as a community-maintained successor that offers improved compatibility, performance, and API standardization [14]. Gymnasium preserves backward compatibility with Gym while enhancing environment stability and observability, making it suitable for modern reinforcement learning frameworks.

Mathematically, an RL environment such as those in Gymnasium can be described as a Markov Decision Process (MDP), represented by the tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ the set of actions, $P(s' \mid s, a)$ the transition probability of moving from state s to s' given action a, $R(s, a)$ the reward function, and $\gamma \in [0, 1]$ the discount factor. The agent interacts with the environment by selecting an action a_t in state s_t, receiving a reward $r_t = R(s_t, a_t)$, and transitioning to a new state s_{t+1} according to P. This iterative loop continues until a terminal state is reached or a predefined horizon is met [15].

2) Tetris as a Reinforcement Learning Benchmark: Tetris is a challenging testbed for reinforcement learning due to its high-dimensional state space, delayed rewards, and stochasticity introduced by random piece generation. Unlike standard environments with immediate feedback, Tetris rewards agents based on long-term strategies such as line clearing and survival duration. The game can also be configured with different action spaces (e.g., discrete rotations and translations) and reward schemes (e.g., score-based or survival-based), making it highly flexible for experimental setups [16].

Early studies on Tetris RL employed heuristic or handcrafted evaluation functions, but recent work has explored deep reinforcement learning approaches, particularly using Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) [17]. These algorithms aim to balance exploration and exploitation in a dynamic and non-stationary environment. However, due to the complex state dependencies and sparse rewards of Tetris, training stable and high-performing agents remains an open challenge. Gymnasium-based Tetris environments facilitate experimentation by providing standardized interfaces for customizing rewards, action constraints, and observation modes, allowing systematic evaluation of algorithmic modifications and parameter tuning [18].

3) Environment Modification and Reward Shaping: Environment modification, including reward shaping, is a common strategy to enhance learning efficiency. Reward shaping involves altering the reward function to provide more frequent feedback or guide the agent toward desired behaviors [19]. In Tetris, shaping rewards by giving intermediate rewards for actions such as clearing lines, avoiding holes, or maintaining a low stack height can significantly affect learning dynamics. Similarly, limiting action spaces to only valid moves can reduce exploration complexity, helping algorithms such as PPO and DQN learn more efficiently within feasible boundaries. However, these modifications must be applied carefully, as excessive shaping may bias the policy toward suboptimal strategies [20].

Overall, Gymnasium and similar frameworks have become indispensable for RL experimentation, enabling consistent and replicable research workflows. Tetris, as implemented within these environments, provides an ideal setting for studying the interplay between algorithmic design, environment modification, and parameter optimization in reinforcement learning.

C. Related Studies

1) AI and Heuristic Methods for Tetris: Early implementations of Tetris-playing agents primarily relied on heuristic-based algorithms rather than deep reinforcement learning. One notable work is the project "Tetris AI — The (Near) Perfect Player," developed by Code My Road [21]. This project used handcrafted heuristics such as aggregate height, number of

holes, bumpiness, and complete lines to evaluate possible piece placements. The agent calculated a weighted sum of these features to score and select the optimal move for each state. Although this approach achieved strong gameplay performance, it heavily depended on manually tuned parameters and lacked adaptability to dynamic or variant game environments. These limitations highlight the need for adaptive learning strategies like reinforcement learning.

2) Reinforcement Learning in Tetris: Several studies have attempted to apply deep reinforcement learning techniques to Tetris. For example, a Stanford CS231N project report [22] explored the use of Deep Q-Networks (DQN) for Tetris, demonstrating that despite the game's high-dimensional and sparse reward space, agents could learn effective placement strategies over time. The study noted, however, that convergence was slow and sensitive to hyperparameter settings, suggesting that experience replay and reward shaping play crucial roles in stabilizing learning in such environments.

3) Experience Replay Strategies in Deep Reinforcement Learning: Experience replay is a key mechanism in off-policy reinforcement learning algorithms like DQN. In traditional replay memory, transitions are sampled uniformly, which helps decorrelate updates but may treat all experiences equally regardless of their significance. To address this, Schaul et al. [23] introduced Prioritized Experience Replay (PER), where transitions with higher temporal-difference (TD) errors are sampled more frequently. This approach increases sample efficiency and accelerates convergence across several Atari benchmark environments. In the context of Tetris, the application of PER could enable agents to prioritize informative transitions, potentially improving learning stability and final performance compared to uniform replay strategies.

D. Research Gap and Study Positioning

While heuristic-based approaches have shown near-perfect gameplay through explicit evaluation and optimization, and Gymnasium provides a robust platform for RL training, there remains limited empirical evaluation of how different experience replay strategies influence the performance of DQN agents in Tetris. This study addresses this gap by comparing standard replay memory and prioritized experience replay within a unified DQN framework using the Tetris-Gymnasium environment.

III. Methodology

A. Environment Picking

The first step in the experimental process involved selecting a suitable environment for training and evaluating the reinforcement learning agent. For this study, the *Tetris-Gymnasium* environment [24] was chosen as the experimental platform due to its standardized design, compatibility with reinforcement learning frameworks, and reproducibility of results.

The Tetris-Gymnasium environment is a modernized implementation of the classic Tetris game, built upon the Gymnasium API (the successor of OpenAI Gym). It provides a consistent interface for observation and action spaces, allowing seamless integration with reinforcement learning algorithms such as Deep Q-Networks (DQN). The environment supports adjustable grid sizes, customizable reward functions, and deterministic seeding for experimental control.

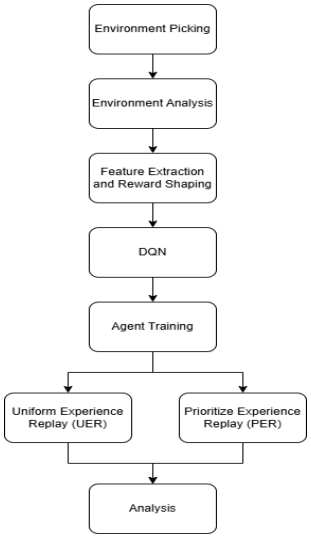

Fig. 1. Training Methodology

Formally, the environment can be represented as a Markov Decision Process (MDP):

$$\mathcal{M} = (S, A, P, R, \gamma) \tag{1}$$

where S denotes the set of states representing the current Tetris grid configuration, A represents the discrete set of possible actions (e.g., rotate, move left, move right, or drop), $P(s'|s, a)$ defines the transition probability from state s to s' after action a, $R(s, a)$ is the scalar reward obtained, and $\gamma \in [0, 1]$ is the discount factor. The agent interacts with the environment iteratively by selecting actions, receiving rewards, and transitioning between states until reaching a terminal condition.

This environment was selected for several reasons:

- **Standardization:** Built on the Gymnasium API, it provides consistent and replicable interaction protocols.
- **Flexibility:** Allows customization of reward functions, grid sizes, and episode constraints to suit different training objectives.
- **Visualization:** Offers both rendered and non-rendered modes for training efficiency and performance evaluation.
- **Benchmarking:** Enables reproducibility across studies using identical environment specifications.

The selection of this environment ensures that subsequent modifications, feature extraction, and algorithmic comparisons are conducted under a controlled and standardized reinforcement learning framework.

B. Environment Analysis

The reinforcement learning experiments in this study were conducted using the *Tetris-Gymnasium* environment [24]. This

environment is designed as a high-fidelity simulation of the classic Tetris game, supporting flexible configurations for agent-based learning and benchmarking. It provides structured interfaces for observation, action, and reward modeling, facilitating integration with value-based and policy-based reinforcement learning algorithms.

1) Environment Description: The environment models the Tetris board as a dynamic grid-based state space, where an agent interacts through discrete actions to control falling tetrominoes. Each game episode consists of sequential time steps, with the environment returning the current observation, reward, and termination flag after each action. The modular architecture enables custom reward shaping, action mappings, and visual rendering, making it well-suited for experimentation and algorithm comparison.

2) Observation Space: The observation space encodes the potential outcomes of each possible action, rather than only the current board state. Specifically, the environment provides:

- **Action-wise Observations:** For each possible action, a corresponding observation is generated.
- **Expanded Dimension:** This adds an extra dimension of size `width × 4` to the observation space.
- **Wrapper Requirement:** Observation wrappers must be applied during environment construction to modify generated observations directly, rather than after a `GroupedActions` wrapper.

This design allows agents to evaluate the effects of different actions in advance, facilitating more informed and strategic decision-making.

3) Action Space: The action space is discrete and encodes all possible placements and rotations of the active tetromino. Specifically:

- **Column & Rotation Choices:** For each column on the board, the agent can choose one of four rotations.
- **Total Actions:** This results in `width × 4` possible actions.
- **Action Encoding:** Action values are interpreted in ascending order of column and rotation. For example:

This setup allows the agent to precisely select both the **placement** and **orientation** for each actions as shown in Table **??**..w

4) Action Space: The action space is discrete and defines the set of valid manipulations available to the agent at each time step. The default configuration is governed by the *ActionsMapping* component, supporting eight distinct actions as shown in Table **??**.

5) Reward Function: The reward function is a configurable component that provides feedback for agent performance. It balances positive reinforcement for efficient gameplay and penalties for undesirable behaviors. A representative mapping is shown in Table I.

Positive rewards encourage line clearing and strategic placement, while negative rewards penalize early termination. This formulation ensures progressive learning by reinforcing long-term strategies.

TABLE I
REWARD MAPPING CONFIGURATION

Name	Value
alife	+0.001
clear_line	+1.0
game_over	-2.0

6) Starting State and Termination: Each episode begins with an empty grid and all the possible block placements. The environment terminates when a new piece cannot be placed due to stack overflow, simulating a game-over condition. This episodic structure allows agents to learn from repeated cycles of play and failure, essential for temporal-difference learning algorithms such as Deep Q-Networks (DQN).

C. Feature Extraction and Reward Shaping

Effective reinforcement learning requires appropriate state representations and reward signals to guide the agent toward optimal behavior. In this study, feature extraction and reward shaping were applied based on prior work by Code My Road [30], which proposed heuristic metrics for Tetris AI agents. The features considered include:

- **Height:** The aggregate height of the columns in the Tetris grid, capturing the vertical distribution of pieces.
- **Holes:** Empty spaces that are covered by at least one block above, representing potential difficulties in clearing lines.
- **Bumpiness:** The sum of the absolute differences in heights between adjacent columns, measuring unevenness of the surface.

These features provide informative cues about the current state of the board, enabling the agent to make decisions that reduce gaps and maintain a flatter surface for easier piece placement.

1) Reward Shaping: To accelerate learning and encourage the agent to clear multiple lines simultaneously, the reward function was defined as follows:

$$\text{reward} = \begin{cases} 1, & \text{alive step} \\ 1 + (10 \times L), & \text{if } L \text{ lines are cleared at once} \end{cases} \tag{2}$$

Here, L represents the number of lines cleared in a single move. This reward function provides a small reward for simply placing a block, promoting survival, and gives larger rewards for clearing lines, particularly multiple lines in one move. Table II summarizes the reward multipliers.

TABLE II
REWARD SHAPING BASED ON LINES CLEARED

Lines Cleared at Once	Reward
0 (alive step)	1
1	11
2	21
3	31
4	41

This reward shaping encourages the agent to prioritize moves that clear multiple lines while maintaining the fundamental objective of surviving as long as possible in the game.

2) Action Masking for Legal Moves: To prevent the agent from selecting illegal actions, we applied *action masking* based on the environment's 'action_mask' provided in the state information. At each timestep, the agent selects from legal actions only. Formally, let $\mathcal{A}_{\text{legal}} \subseteq A$ denote the set of legal actions in the current state. The action selection policy $\pi(a|s)$ is defined as:

$$a_t = \begin{cases} \text{random choice from } \mathcal{A}_{\text{legal}}, & \text{with probability } \epsilon \\ \arg\max_{a \in \mathcal{A}_{\text{legal}}} Q(s_t, a; \theta), & \text{with probability } 1 - \epsilon \end{cases} \tag{3}$$

Here, ϵ is the exploration rate, and $Q(s, a; \theta)$ represents the predicted Q-values for the current state. Any illegal action is assigned a masked value of $-\infty$ to ensure it is never selected. This approach reduces the likelihood of invalid moves and improves the efficiency of the learning process by constraining the agent to feasible actions.

D. Deep Q-Network (DQN) and Agent Training with PER and UER

1) DQN Architecture and Training Procedure: The Deep Q-Network (DQN) used in this study is a fully connected neural network that approximates the action-value function $Q(s, a)$. The input features for the network include the height of columns, number of holes, bumpiness of the Tetris grid, and a block-placement indicator (alive step). The network architecture consists of three linear layers with ReLU activations:

$$\text{Input } (4) \rightarrow \text{Linear(4,64)} \rightarrow \text{ReLU} \rightarrow \text{Linear(64,64)} \rightarrow \text{ReLU} \rightarrow \text{Linear(64,1)} \rightarrow \text{Output Q-values}$$

All linear layers are initialized using Xavier uniform initialization, and biases are set to zero to stabilize training.

2) Experience Replay: UER and PER: The agent's transitions (s_t, a_t, r_t, s_{t+1}) are stored in a replay buffer. Two replay strategies are considered:

- **Uniform Experience Replay (UER):** Transitions are sampled uniformly at random from the buffer.
- **Prioritized Experience Replay (PER) [24]:** Transitions are sampled according to their TD-error δ_i with probability:
$$P(i) = \frac{(|\delta_i| + \epsilon)^\alpha}{\sum_k (|\delta_k| + \epsilon)^\alpha}$$
where α controls the prioritization. Importance-sampling weights correct the bias introduced by prioritization.

Both replay strategies are trained separately using the same DQN architecture and identical hyperparameters. This setup ensures that performance differences are due solely to the replay strategy.

3) Reward Shaping: Reward shaping is applied to both UER and PER to encourage clearing lines while also rewarding survival (alive steps).

4) Training Procedure: The DQN agent is trained as follows:

1) Initialize the DQN network parameters θ and the replay buffer (UER or PER) with the following hyperparameters:
   - Episodes: 3000
   - Batch size: 512
   - Discount factor (γ): 0.99
   - Learning rate: 1e-4
   - Epsilon-greedy: $\epsilon_{\text{initial}} = 1.0, \epsilon_{\text{min}} = 0.05, \epsilon_{\text{decay}} = 0.9985$
   - Target network update frequency: 1000 steps
   - Replay memory size: 30,000 transitions
2) For each episode:
   a) **Feature Extraction:** Extract the four features from the current state.
   b) **Action Selection with Masking:** Mask illegal actions and select a_t using the ϵ-greedy policy.
   c) **Environment Interaction:** Execute the action, observe reward r_t with shaping, and transition to s_{t+1}.
   d) Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer.
   e) Sample a minibatch from the buffer and perform gradient descent on the TD-loss:
$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - Q(s_i, a_i; \theta)\right)^2 \qquad y_i = r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) \tag{4}$$
   f) Periodically update the target network: $\theta^- \leftarrow \theta$.
3) Repeat until convergence or maximum episodes.

E. Evaluation Metrics

To assess the performance of the DQN agent with both Uniform Experience Replay (UER) and Prioritized Experience Replay (PER), several metrics are considered. These metrics provide insights into both the learning efficiency and gameplay effectiveness of the agent.

1) TD-Loss: The temporal-difference (TD) loss is monitored during training of the DQN agent with Prioritized Experience Replay (PER). It measures the discrepancy between the predicted Q-values and the target Q-values, and indicates how well the network is learning from prioritized transitions. It is computed as:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - Q(s_i, a_i; \theta)\right)^2, \qquad y_i = r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) \tag{5}$$

A lower TD-loss suggests that the PER agent is learning more accurate Q-value estimates and effectively updating from high-priority transitions.

2) Average Reward: The average reward is calculated over 3,000 training episodes for both UER and PER agents. It provides a measure of the overall performance of the agent, reflecting both survival and line-clearing efficiency. Formally, the average reward per episode is computed as:

$$\bar{R} = \frac{1}{E} \sum_{e=1}^{E} R_e, \tag{6}$$

where E is the number of episodes and R_e is the cumulative reward obtained in episode e.

3) Total Lines Cleared: This metric counts the cumulative number of lines cleared by the agent across all episodes. It highlights the agent's ability to achieve the primary objective in Tetris, i.e., clearing lines efficiently.

4) Maximum Lines in an Episode: The maximum number of lines cleared in a single episode captures the agent's peak performance and its ability to execute high-scoring sequences. This is particularly relevant in evaluating the effectiveness of PER, which prioritizes rare high-reward transitions.

Together, these metrics provide a comprehensive evaluation framework, enabling a comparison of UER and PER in terms of learning stability, cumulative performance, and peak gameplay ability.

IV. Results and Discussion

A. Comparison of UER and PER

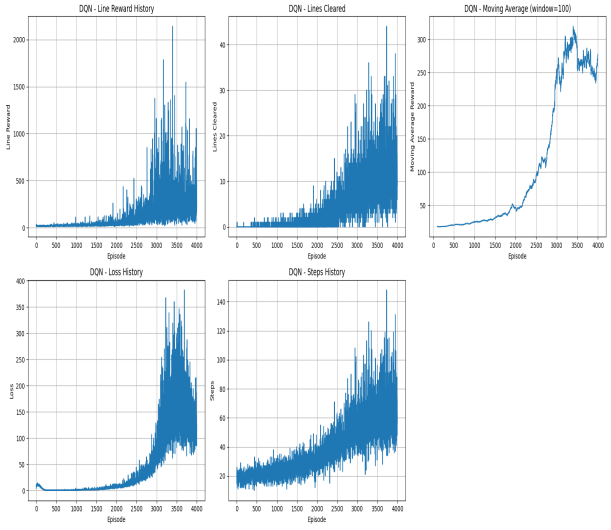

Fig. 2. Training performance of the DQN agent using Uniform Experience Replay (UER).

The DQN agent trained with **Uniform Experience Replay (UER)** outperformed the agent trained with **Prioritized Experience Replay (PER)** across most metrics. Over 4,000 episodes, UER achieved an average reward of 109.68, cleared a total of 15,809 lines, and averaged 3.95 lines per episode. In comparison, PER achieved an average reward of 78.17, cleared 8,261 lines, and averaged 2.07 lines per episode. Both

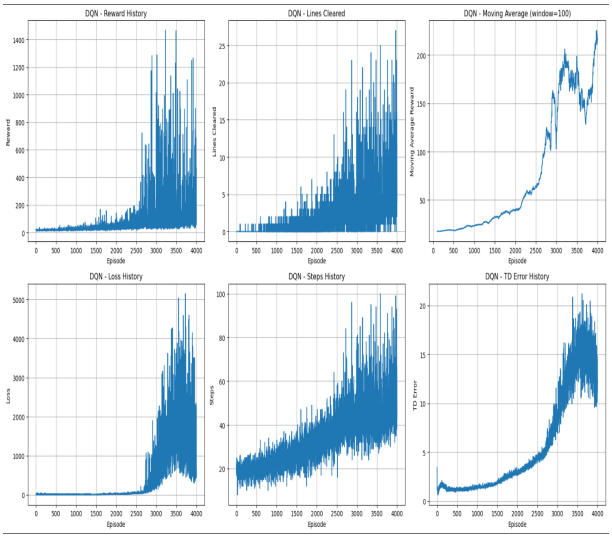

Fig. 3. Training performance of the DQN agent using Prioritized Experience Replay (PER).

strategies reached a maximum of 34 lines cleared in a single episode.

The superior performance of UER can be attributed to several factors. The reward shaping scheme provides substantial rewards for clearing multiple lines simultaneously. PER, which prioritizes transitions based on TD-error magnitude, may overemphasize these rare events, introducing learning instability. Additionally, the large batch size (512) and buffer size (30,000) allow UER to sample important transitions more consistently, while uniform sampling smooths learning by averaging over all transitions, mitigating noise amplified by PER's prioritization.

These results indicate that, under the current Tetris environment, reward shaping, and network configuration, **UER provides more stable and effective learning than PER**, achieving higher overall rewards and more consistent line-clearing performance.

TABLE III
Training Performance of DQN with Uniform Experience Replay (UER) and Prioritized Experience Replay (PER)

Metric	UER	PER
Total Episodes	4000	4000
Average Steps/Episode	37.78	32.96
Average Reward/Episode	109.68	78.17
Best Reward	2140.00	1466.00
Total Lines Cleared	15,809	8,261
Average Lines/Episode	3.95	2.07
Max Lines in Single Episode	44	27

V. Conclusion

The evaluation of the DQN agent in Tetris demonstrates that Uniform Experience Replay (UER) outperforms Prioritized Experience Replay (PER) under the current experimental setup. UER achieved higher average rewards and more lines cleared per episode, indicating more stable and consistent

learning. The observed advantage of UER is likely due to its uniform sampling, which mitigates the instability caused by PER's overemphasis on rare high-TD-error transitions, especially in combination with the large batch and buffer sizes used. These results suggest that, in environments like Tetris with reward shaping favoring multi-line clears, UER provides a more reliable training strategy, offering smoother convergence and more consistent performance than PER.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[3] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, pp. 2094–2100, 2016.

[4] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning, and teaching," *Machine Learning*, vol. 8, no. 3–4, pp. 293–321, 1992.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[6] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 3215–3222.

[7] A. F. Karakovskiy and J. Togelius, "The Tetris benchmark for reinforcement learning," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 85–92.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," J. Artif. Intell. Res., vol. 4, pp. 237–285, 1996.

[9] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 8, pp. 1798–1828, 2013.

[10] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," Proc. 4th Connect. Models Learn. Syst., 1993.

[11] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[13] Gymnasium Community, "Gymnasium: A toolkit for developing and comparing reinforcement learning algorithms," 2022. [Online]. Available: https://gymnasium.farama.org

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[15] F. Fahey, "Tetris as a benchmark for reinforcement learning algorithms," *IEEE Trans. Games*, vol. 12, no. 3, pp. 245–257, 2020.

[16] M. L. Matignon, N. Fort-Piat, and D. Laurent, "Deep reinforcement learning for Tetris: Challenges and insights," in *Proc. Int. Conf. on Machine Learning and Applications (ICMLA)*, 2019, pp. 1121–1128

[17] Gymnasium-Tetris Developers, "Tetris environment for reinforcement learning," 2023. [Online]. Available: https://gymnasium.farama.org/environments/tetris

[18] A. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. 16th Int. Conf. on Machine Learning (ICML)*, 1999, pp. 278–287.

[19] J. Brys, A. Harutyunyan, M. Vrancx, A. Nowé, and M. E. Taylor, "Policy transfer using reward shaping," in *Proc. Int. Conf. on Adaptive and Learning Agents (ALA)*, 2015, pp. 27–41.

[20] C. My Road, "Tetris AI – The (Near) Perfect Player," *Code My Road Blog*, 2013. [Online]. Available: https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player

[21] S. Yan and R. Hoang, "Deep Reinforcement Learning for Tetris," Stanford CS231N Project Report, Stanford University, 2016. [Online]. Available: https://cs231n.stanford.edu/reports/2016/pdfs/121$_Report.pdf$

[22] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv preprint arXiv:1511.05952*, 2015.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[24] M. Weichart, "Tetris-Gymnasium: A Modular RL Environment for Tetris," GitHub repository, 2023. [Online]. Available: https://github.com/Max-We/Tetris-Gymnasium

[25] C. My Road, "Tetris AI – The (Near) Perfect Player," *Code My Road Blog*, 2013. [Online]. Available: https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player