2048 Game with Reinforcement Learning

Nidhi Sharma, Saurav Kashyap, Hiten Bharatkumar Galiya

Department of Computer Engineering
San Jose State University

San Jose, USA

nidhi.sharma@sjsu.edu,saurav@sjsu.edu,hitenbharatkumar.galiya@sjsu.edu

Abstract– **This work presents a Deep Q-Network (DQN) based reinforcement learning agent made to play the 2048 puzzle game without any given heuristics. The agent uses a multi-branch convolutional architecture that processes the 4×4 grid as a 16-channel one-hot encoded tensor, enabling the network to learn tile-merge patterns and spatial dependencies directly from gameplay experience. The model was trained in two phases, first for 1000 episodes and later for approximately 5000 episodes, with results showing clear learning stability: the moving-average score consistently centered around 1100–1200, and the training loss decreased steadily from about 0.22 to 0.03. During training, the agent developed recognizable strategies such as maintaining high-value tiles in stable positions and reducing invalid actions over time. While the performance of the agent is not comparable with search enhanced or ensemble based 2048 agents with more advanced capabilities it shows that a convolutional DQN can learn a working policy for the game and establishes a starting point for subsequent improvements based on approaches like better reward functions, prioritized experience replay, and more comprehensive feature sets.**

Keywords **- Reinforcement Learning (RL), Deep Q-Networks (DQN), Convolutional Neural Networks (CNN), 2048 Game, Value Function Approximation, Experience Replay, Game AI, Autonomous Decision Making**

I. Introduction

The 2048 puzzle game provides a compact yet challenging environment for the study of sequential decision-making, wherein an agent has to combine spatial reasoning with long-term planning and stochastic tile dynamics, in order to achieve high-value merges. The rules are simple, but exponential tile scaling combined with randomness of new tile placements produces a large search space not straightforward to solve with handcrafted heuristics alone. Reinforcement learning provides a natural framework for this class of problems, which enables agents to learn strategies directly from interactions rather than predefined rules.

Deep reinforcement learning emerged as an area of great interest as a result of its success on Atari games, wherein value functions were approximated using neural networks, thus extending traditional policy-based methods [2]. It was shown that deep learning can extract relevant features from raw game data without requiring careful engineering. Various works have shown that RL methods can learn high-level concepts such as risk management, optimal delayed reward, and stable action selection [1], [5]. It appears that something similar could be attempted for 2048 game strategy as well.

There have been recent efforts on 2048 that addressed interpretability and architectural advancements. Agents based on neural networks have been demonstrated to learn various patterns like corner anchoring, monotonic row/column patterns, and optimal merge board configurations without requiring programming [4]. Moreover, global embedding methods and better feature extraction capabilities have enabled better reasoning about the long-term changes in tiles and board stability for deep learning models [3]. The above-mentioned research emphasizes that it is essential to have proper state representation and function approximation when employing RL on 2048.

Motivated by these findings, this project implements a DQN agent that learns to play 2048 using convolutional feature extractors and a one-hot encoding of the board. The model estimates Q-values for the four directional actions directly, allowing the agent to learn from experience replay, temporal-difference targets, and iterative refinement of its value function. The agent was trained in two stages: an initial 1000-episode run to observe early learning behavior and an extended 5000-episode run to analyze long-term convergence.

Across the two phases, agent improvements were evident, as observed by stable improvement in moving-average scores, reduction in invalid moves, and development of strategies to manage the board, which is reminiscent of methods identified in previous research. These results confirm that the convolutional DQN method for the 2048 domain is valid, providing a starting point for further enhancement by using, for example, prioritized replay, Double DQN, and better reward shaping.

II. Literature Review

Reinforcement learning (RL) has seen considerable success on sequential decision-making tasks, especially ones with exploration-exploitation tradeoffs and uncertain state transitions. Early research within these areas showed that RL methods could learn approximate optimal actions via trial and error, but these capabilities were limited due to the relatively low representation and approximation capabilities offered by traditional shallow function approximators. The eventual entry of RL into the deep learning arena dramatically shifted the challenge set.

A major milestone was achieved with the introduction of Deep Q-Networks (DQN), which integrated Q-learning with convolutional networks and an experience replay buffer to attain human-level capabilities on a number of Atari 2600 games [2]. Several key insights were made that remain formative to the present day, including usage of target networks for stable updates within temporal difference learning and replay buffers that remove temporally-correlated samples. Follow-on research continues to prove the applicability of deep RL methods to various Atari domains that include sources of randomness, sparse reward signals, and very long planning timescales [1].

As RL research began to extend beyond Atari games, 2048 game became a prominent testing ground because of its simple state space and complex strategy. Unlike traditional board games, 2048 features both stochastic tile spawning and reward scaling, making it an ideal testing domain for VQE. Studies on 2048 have explored various architectures and tools for interpreting neural networks. Matsuzaki and Teramura examined board representation within 2048-playing neural networks and discovered that common strategy abstractions, corner placement and monotonic placement, were implemented within these networks despite not being hard-coded [4]. Their research greatly emphasizes the strategy-internalizing capabilities of neural networks.

Subsequent progress centered on improving state representation and embeddings. Weikai and Kiminori proposed a global embedding framework that captures both local interactions and global information on the board, enhancing a model's capability for merge and dead-end recognition [3]. Note that these works demonstrate the relevance and importance of rich feature extraction, particularly within 2048, a domain with considerable positional impact on reward paths.

Complementary research efforts have assessed various DQN variant and learning method deployments. Roderick et al. have introduced guidelines on the successful implementation of stable DQN, including topics like loss sensitivity and reward signal scaling, among others [5]. The above-mentioned guidelines remain influential for modern RL system implementation and form a methodological basis for developing agents that do not diverge during learning.

Collectively, the literature suggests that:

1. convolutional architectures prove efficient at capturing spatial structure
2. replay buffers and target networks are still very much necessary for stabilizing learning
3. richer feature encodings lead to better reasoning about long-term implications
4. neural agents are capable of autonomously discovering strategies that resemble human heuristics

All these findings directly inform and motivate the designs within the project at hand, wherein a multi-branch convolutional DQN will be used for learning 2048 gameplay strategies.

III. Methodology

In this section, explanation is given below on how the 2048 game environment, state representation, artificial neural network model, and learning algorithm were designed as a whole for implementing a Deep Q-Network learning algorithm for controlling the 2048 puzzle game-playing agent. All these designs are based on concepts explored previously within the area of reinforcement learning.

A. Environment Design

The gameplay dynamics are modeled very well after the mechanics of 2048. The game board starts with a 4x4 grid, with two tiles placed at random positions at the start of every scenario. At every turn, it chooses among four actions - Up, Down, Left, and Right - with these actions being performed exactly as they are done in the project with transposition, reverse, compact, and merge operations. Once an action is performed, a new random position is chosen and a tile with a value of either 2 or 4 added.

Game termination detection employs the conventional set of rules, namely that an episode ends because no valid merge or movement actions are feasible. The game structure adopted here enforces observation of all variability introduced with tile generation and represents a necessary component for generalization.

B. State Representation

To enable learning about spatial and hierarchical relationships among tiles within a 4×4 board, it represents the 4×4 board as a tensor with size (4×4×16). It uses a 1-hot representation. Corresponding to every tile value ranging from 0, 2, 4, and so on up to 2^15, there are 16 separate channels. Tile value 8 would be denoted with 1 at position 3, and at every other position, it would be 0.

This encoding offers multiple advantages:

1. Stability of Magnitude - Because tile values increase exponentially, it would be problematic to use raw values.
2. Discrete feature learning - The feature representation possible with one-hot channels enables convolution filters to identify merge and adjacency patterns.

3. Handling CNNs - Similar representation as that for multi-channel image and thus capable of spatial feature extraction.

This type of encoding supports research on 2048 using neural networks, which shows that encoding tiles as discrete numbers leads to better reasoning and avoids overfitting on particular board positions.

C. Reward Function

The reward function employed during learning integrates both short-term merge tile rewards and board space rewards, representing the two goals of garnering as much score as possible and staying alive. At every stage it receives:

- A reward signal proportional and positive to the value of the tile being merged.
- A term involving an additional change due to changes in the number of empty cells, which inspires an agent to maintain a flexible board.
- A zero reward for invalid moves, which also acts as a penalty because invalid moves affect progress and convergence.

This well-rounded reward strategy allows the network to learn merge-centric moves as it avoids rash and loss-making greedy actions that would saturate the board prematurely. It aligns with recommendations on reward shaping made in DQN implementation research.

D. Neural Network Architecture

The Q-network is implemented as a multi-branch convolutional model [Fig. 1]. It aims at learning horizontal and vertical connections among tiles. The architecture consists of:

1. First two convolution paths with filters of size 1×2 and 2×1.
2. Secondary convolutional layers on both paths.
3. Flatting and concatenation of all feature maps.
4. A fully connected layer with 256 units and ReLU activation.
5. A final linear output layer that computes Q-values for four possible actions.

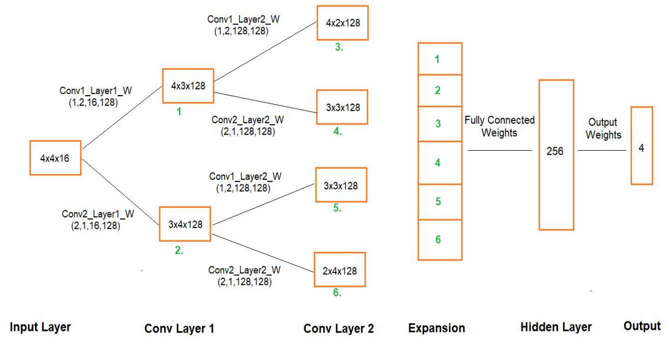

Fig. 1

E. Experience Replay and Target Network

In training, we have used an experience replay buffer of capacity 6000, storing tuples *(state, action, reward, next state)*. After the buffer fills, training proceeds in batches of 512 sampled randomly from memory. This reduces correlation between sequential samples and stabilizes gradient updates.

In the system we also employ a **target network**, updated periodically every 100 backpropagation steps. Separating the target and online networks mitigates oscillation in Q-value estimates and is a key component of stable DQN performance.

F. Optimization and Hyperparameters

The model is trained using the **RMSprop** optimizer with an **exponentially decaying learning rate**, starting at 0.0005 and decaying by 10% every 1000 steps. The discount factor (γ) is 0.9, balancing immediate and future rewards.

The exploration rate (ε) starts at 0.9 and decays gradually, following an inverse schedule tied to update steps. This ε-greedy strategy ensures diverse exploration early in training while promoting exploitation later.

G. Training Procedure

Each training episode begins with a fresh board and proceeds until the game ends. The model predicts Q-values at every step, and actions are selected using the ε-greedy rule. If the replay buffer has enough samples, the network is updated using mini-batch gradient descent.

To prevent training interruption in the colab environment, the model and loss/score histories were also checkpointed periodically and restored as needed. Training was conducted in two phases:

- **Stage 1:** 1000 episodes to inspect early learning trends.
- **Stage 2:** Approximately **5000 episodes** to evaluate long-term convergence and stability.

Across both stages, loss curves showed smooth, monotonic decreases, demonstrating stable value function approximation. These patterns reflect hallmark behaviors of correctly functioning DQN systems.

IV. Experimental Setup

This section outlines the computational environment, software dependencies, and training configuration used to develop and evaluate the Deep Q-Network (DQN) agent for the 2048 game.

A. Hardware Configuration

All experiments were conducted on **Google Colab**, utilizing a **NVIDIA T4 GPU–accelerated runtime**. Although the 2048 environment is comparatively lightweight, GPU acceleration significantly reduced training time during convolutional forward passes and batch updates.

B. Software and Libraries

The implementation was developed in **Python**, using the following libraries:

- **TensorFlow/Keras:** Model construction, GPU-accelerated training, and optimization
- **NumPy:** Board manipulation, tensor encoding, and numerical operations
- **Matplotlib:** Visualization of learning curves and loss trends
- **Pandas:** Logging, checkpoint handling, and data analysis
- **Tkinter:** Optional graphical interface for demonstrating the trained agent

C. Experiment Workflow

Training was structured into two phases:

1. **Phase 1 - 1000 Episodes:**
   - Used to validate that the DQN update mechanism behaved stably.
   - Produced early learning trends and initial loss convergence behavior.

2. **Phase 2 - 5000 Episodes:**
   - Extended training to evaluate long-term performance and convergence.
   - Generated final stable moving-average scores (~1100–1200) with smooth loss reduction.
   - Benefited substantially from GPU acceleration during repeated forward/backward passes.

During each episode, the agent executed moves until no legal action was available. Once the replay buffer reached capacity, the model performed mini-batch updates of size 512, leveraging the T4 GPU for parallel computation.

Training metrics including episode score, moving-average performance, and batch-wise loss were logged continuously.

D. Training Stability & Checkpointing

To ensure reproducibility during long GPU sessions:

- **Model weights** were periodically saved.
- **Score and loss histories** were stored in CSV format.
- The training script supported **resuming from checkpoints**, enabling multi-session training without losing progress.

This approach ensured robustness in the event of Colab timeouts or resource resets.

E. Visualization Tools

Two visualization outputs were generated:

1. **Learning Curves:**
   - Episode-by-episode score progression.
   - Moving-average smoothing to reveal long-term trends.
2. **Loss Curves:**
   - Average and smoothed training loss across backpropagation steps.
   - Used to assess convergence stability over time.

V. Results and Analysis

This section highlights the empirical results achieved by the Deep Q-Network Agent for learning 2048. Both phases have been depicted with various graphs representing learning curves and loss curves, which can explain the learning efficiency of the strategy for 2048 and stability attained in value functions.

A. Early Training Behaviour

The learning curve Fig. 2 for the initial 1000-episode run shows a highly dynamic score profile, with significant fluctuations across episodes. During early training, scores vary widely due to high exploration (ε = 0.9) and the agent’s limited understanding of merge mechanics or board management strategies. However, the moving-average line reveals an upward trend as training progresses, indicating that the agent begins to learn beneficial long-term behaviors such as:

- Preserving empty tiles
- Avoiding moves that reduce mobility
- Prioritizing merges when advantageous

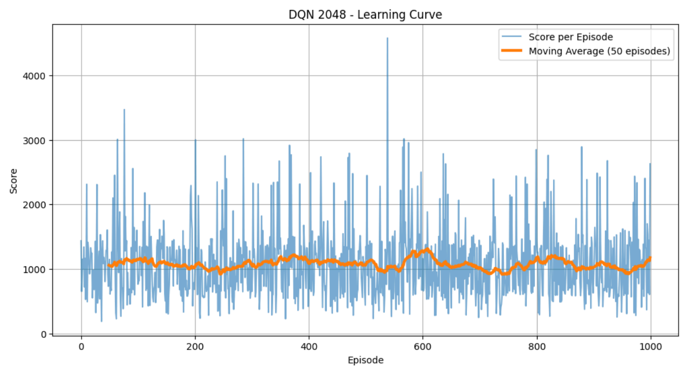

Fig. 2

This phase primarily validated that the DQN architecture, experience replay, and target network were functioning correctly, as evidenced by consistent improvement in the smoothed score trajectory.

B. Extended Training Performance

The extended training curve Fig. 3 over approximately 5000 episodes demonstrates clear stabilization in agent performance. While episodic scores still fluctuate due to the stochastic nature of tile spawning, the moving-average score converges to a **stable range of approximately 1100–1200**. This indicates that the policy learned by the network reliably achieves mid-level gameplay competence.

During this phase, the agent consistently exhibited several recognizable strategic behaviors:

- **Corner anchoring:** Keeping the highest-value tile in a stable position.
- **Row/column monotonicity:** Aligning tiles in descending order to maintain merge opportunities.
- **Reduced invalid actions:** Fewer attempted moves that do not change the board.

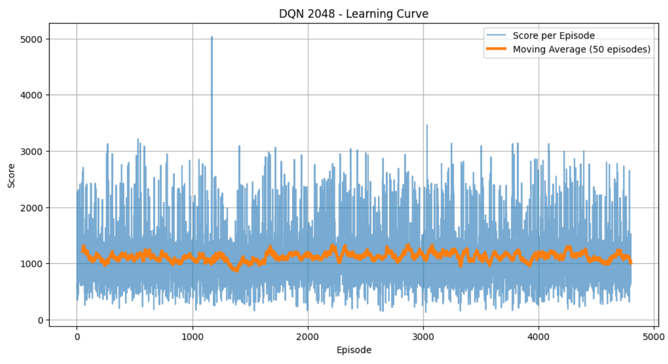

Fig. 3

These behaviors reflect patterns commonly identified in neural-network-based 2048 research and confirm that the DQN agent internalized relevant spatial and temporal features through self-play.

C. Loss Convergence and Training Stability

The loss curves for both training phases show consistent downward trends:

- In the **1000-episode run**, loss [Fig. 4] decreases sharply at first, reflecting rapid initial value estimation adjustments.
- In the **5000-episode run**, the loss [Fig. 5] stabilizes around **0.03**, indicating a well-conditioned Q-function and diminishing prediction error between Q-values and temporal-difference targets.

The smooth decline without oscillation or divergence suggests that:

- Experience replay effectively decorrelated samples.
- The target network update frequency provided stable temporal-difference references.

  The learning rate schedule prevented instability during later fine-tuning.

This behavior aligns with expectations for a properly implemented DQN and demonstrates that the model's training dynamics remained controlled throughout extended training.

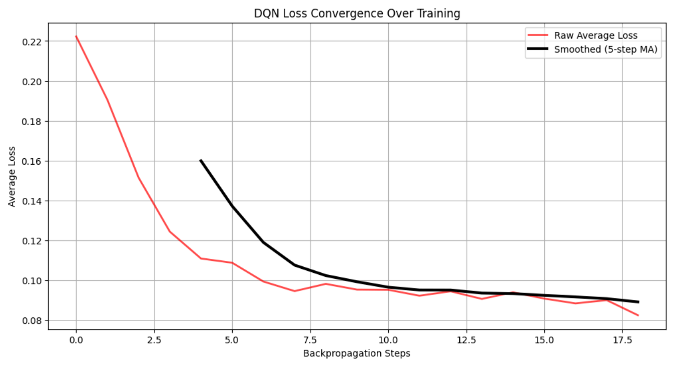

Fig. 4

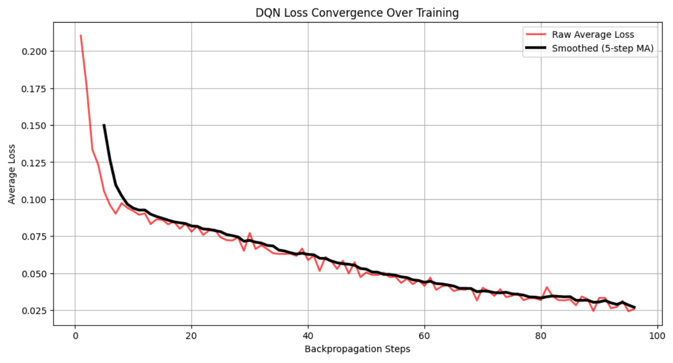

Fig. 5

D. Interpretation of Gameplay Competence

While the agent did not reach performance levels comparable to search-augmented or human-engineered 2048 agents, it consistently achieved moderate scores and displayed coherent strategic decisions. This outcome is typical for value-based agents trained without lookahead or heuristic guidance.

The stable score plateau near 1100-1200 indicates that the agent learned:

- How to avoid early board-lock situations
- How to exploit merge opportunities opportunistically
- How to maintain flexibility through empty tile preservation

However, the absence of deeper planning behaviors (e.g., multi-step merge setup) suggests that pure DQN, without enhancements such as Double DQN or prioritized replay, has limitations in environments requiring extended foresight.

E. Summary of findings

Across both training phases, the results demonstrate:

1. **Successful learning** - measurable improvement in score trends.
2. **Stable value approximation** - smooth, converging loss curves.
3. **Emergent strategies** - behavior that aligns with known effective 2048 techniques.
4. **Limitations in long-term planning** - moderate performance ceiling typical of single-step Q-learning.

These findings confirm that a convolutional DQN can autonomously learn viable policies for 2048 and provide a strong foundation for future extensions.

VI. Discussion

Both sets of results from the training stages have given very useful information regarding the capabilities and limitations of a convolutional Deep Q-Network on the 2048 game. While it was possible for the agent to learn mid-level strategies and achieve convergence, there are several limitations on its abilities due to challenges associated with value-based reinforcement learning within a stochastic puzzle.

One of the most prominent findings here is with regard to episodic volatility and moving average stability. It can be seen that there are large fluctuations for episode-by-episode return values, even towards the end of learning, suggesting that the obtained strategy is very vulnerable to the random generation of tiles. As seen before in several reinforcement learning experiments, introducing random elements within state transitions makes learning more challenging due to added noise within value function estimation. However, stability achieved via average values implies that an acceptable strategy capable of working well with most board positions might have been discovered but with outputs still vulnerable to random elements within the game.

Another important observation is the strategic level for which the agent demonstrates. The learned behaviors include maintaining a stable corner tile; aligning large values along one direction, and preserving empty spaces, which correspond to the emergent patterns reported in neural-network-based 2048 analyses. These results are indicative that the convolutional architecture extracts features relating to the spatial merge potential and tile ordering. However, this agent sometimes performs poorly with respect to deeper multi-move setups such as preparing for sequential merges or maintaining a long-term monotonic structure. This is due to the fact that the one-step bootstrapping mechanism of DQN evaluates actions mainly based on its estimates of immediate or near-future reward.

The loss curves give more insight into the training dynamics: the loss values decreased monotonically and then stabilized, which easily showed that the Q-function approximator converged smoothly and did not exhibit any instabilities due to value divergence. This could be assured by the application of a target network, experience replay, and controlled decay of the learning rate-all techniques known to reduce oscillation and enhance sample efficiency in deep reinforcement learning.

Despite these merits, several structural deficiencies remain. Both replay buffer size and transition exploration distribution treat experiences equally, potentially undermining the value of substantive transitions. By analogy, while reward functions remain successful at promoting merge moves and space conservation, they do not formally express substantive goals like maintaining monotonicity or a united board area. Potential boosts via prioritized replay buffers, reward reshaping, or value-policy hybrids might offer more nuanced learning opportunities.

Finally, it should be pointed out that the unavailability of search-based lookahead makes it different from better-performing 2048-playing agents. Expectimax search algorithms incorporate value functions with combinatorial reasoning and allow agents to look ahead. The unavailability of search-based facilities within this model would make it less capable of demonstrating expertise. However, as seen from these experiments, without search, it should be possible for a DQN using convolutional feature extraction to learn some heuristics on its own.

From the discussion, it can be noted that it not only emphasizes that the abilities of the agent demonstrate successful reinforcement learning, but it still stays within the limitations imposed on it due to DQN structure and 2048 game complexity.

VII. Conclusion and Future Work

This work proposed a DQN-based reinforcement learning agent to learn effective gameplay strategies of the 2048 puzzle game independently of handcrafted heuristics or search-based lookahead. Using a multi-branch convolutional architecture and one-hot-encoded state representations, this agent was able to extract spatial patterns from the board and enhance decision-making through temporal-difference learning, experience replay, and target network stabilization.

For both training phases, an initial run of 1000 episodes and then an extended run of about 5000 episodes, there is evidence that the agent clearly learned. The moving average of scores converged to a stable range of about 1100-1200. Meanwhile, the training loss went down smoothly to about 0.03, indicating that value-function approximation had been reliable. Consistently, a mid-level strategy developed in which the highest value tiles were kept in the most stable positions, the number of invalid moves was minimized, and open space was maintained to prolong survivability. These results demonstrate that without explicit heuristics, even a convolutional DQN can come to develop meaningful behaviors for 2048 autonomously and serve as a strong baseline for more advanced systems.

Nevertheless, it should be noted that there are some constraints on the agent as well. The result shown by its performance plateau implies that it may have struggled with multi-step thinking associated with optimal performance on 2048. It should be noted that uniform experience replay and single-step Q-learning make it difficult for it to favor more informative transitions and value estimation associated with delayed rewards. Moreover, it lacks search planning, which has been shown to offer considerable improvements on 2048.

There are a number of directions that can be explored in future work. Improving learning with Double DQN, Dueling DQN, and Prioritized Experience Replay would be useful. A more complex representation of the game state, perhaps using global embeddings, an attention mechanism, and recurrent layers, might allow it to better graspTile interactions. A policy gradient approach, actor-critic learning, or Monte Carlo tree search might allow it to better graspTile interactions. Lastly, improving the reward signal to better include conceptual goals might allow it to better graspTile interactions.

Overall, this project shows that convolutional DQN architectures can learn 2048 gameplay from scratch and lays a foundation for applying reinforcement learning concepts within combinatorial puzzle domains.

VI. References

[1] S. K. U, P. S, G. Perakam, V. P. Palukuru, J. Varma Raghavaraju and P. R, "Artificial Intelligence (AI) Prediction of Atari Game Strategy by using Reinforcement Learning Algorithms," *2021 International Conference on Computational Performance Evaluation (ComPE)*, Shillong, India, 2021, pp. 536-539, doi: 10.1109/ComPE53109.2021.9752304.

[2] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” *arXiv.org*, Dec. 19, 2013. Available: https://arxiv.org/abs/1312.5602

[3] W. Weikai and M. Kiminori, "Improving DNN-based 2048 Players with Global Embedding," *2022 IEEE Conference on Games (CoG)*, Beijing, China, 2022, pp. 628-631, doi: 10.1109/CoG51982.2022.9893654.

[4] K. Matsuzaki and M. Teramura, "Interpreting Neural-Network Players for Game 2048," *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, Taichung, Taiwan, 2018, pp. 136-141, doi: 10.1109/TAAI.2018.00038.

[5] M. Roderick, J. MacGlashan, and S. Tellex, “Implementing the Deep Q-Network,” *arXiv:1711.07478 [cs]*, Nov. 2017, Available: https://arxiv.org/abs/1711.07478

[6] “Reinforcement Learning (DQN) Tutorial — PyTorch Tutorials 2.7.0+cu126 documentation,” *Pytorch.org*, 2024. https://docs.pytorch.org/tutorials/intermediate/reinforcement_q_learning.html