Extended Abstract

Motivation The game 2048 is a sparse-reward environment that mirrors challenges found in many applications such as robotics and budgeting. The infrequent reward structure makes it difficult for agents to learn actions that eventually contribute to long-term success. SOTA 2048 agents often rely on heuristics, simulations, and brute-force search, which are computationally expensive and require millions of trials. To improve sample efficency, developing self-sufficient deep RL-based approaches for 2048 can reveal more general principles for learning in environments with delayed rewards, limited feedback, and constrained action spaces.

Method For our baseline, we implemented a Deep Q-Network that we used as a training framework throughout our experiments. The baseline agent operated on a flattened 4×4 board and was trained using ε-greedy exploration, experience replay, and a target network. Since flattening the board gives the agent no useful signals about tile arrangements and potential merges, we introduced a CNN encoder to produce feature maps that capture local spatial patterns on the board. We then introduced auxiliary tasks to further enrich the feature maps; one head predicted the legality of each action, and another predicted the future maximum-value tile after taking an action. The tasks aimed to help the action learn how to efficiently play the game without making sporadic moves, and look towards long-term success, respectively. To more explicitly guide learning, we incorporated reward shaping, which consisted of giving a penalty for illegal moves and a bonus that incentivized keeping the number of tiles on the board to a minimum to make more space for future merges. For our evaluation, we conducted ablation studies to isolate the impacts of each method and tracked key metrics such as illegal moves, losses, and reward bonuses/penalties.

Implementation Our models were built using PyTorch and integrated with a custom Gym environment based on `gymnasium_2048`, which also offers live visualizations of the game and out-of-the-box metrics such as the maximum-value tile reached, total score, and whether an action was an illegal move. We modularized the agent's network architecture and training configurations, making it easy to extend with the CNN encoder, auxiliary heads, reward shaping, and hyperparameter optimization.

Results The addition of representation learning led to minor improvements over the baseline, which showed that the agent was randomly making illegal moves and failing to improve its scores from the start of training. The CNN encoder most significantly reduced the number of illegal moves, and the auxiliary tasks and reward shaping stabilized training and encouraged longer-term planning, resulting in the full model being able to reach higher tile values more consistently. However, overall performance hit a ceiling, with the agent rarely surpassing tile 512, highlighting the ongoing difficulty of mastering 2048 through model-free RL alone. Qualitatively, we experience the agent to perform moderately on learning a promising strategy. The agent tends to favor a balance between merging and creating new tiles, whereas merging more often would be a better strategy.

Discussion While the additions of CNN-based representation learning, auxiliary tasks, and reward shaping improved over a standard DQN and significantly reduced illegal moves, the agent was still short-sighted likely due to focusing on immediate rewards (i.e. merging tiles resulting in a slight score increase). The implication is that the agent fails to set up multi-step, planned strategies that yield delayed but higher returns. It struggled to realize strategies such as keeping high-value tiles in corners or maintaining monotonic rows or columns to future high-value merges. Thus, the results suggest that while intermediate learning signals can nudge the agent toward more structured behaviors, overcoming the difficulties from sparse reward environments likely requires planning and more adaptive exploration above ε-greedy policies.

Conclusion Through this project, we conclude that incorporating representation learning and reward shaping to a DQN improves an RL agent's ability to navigate the board, but still achieves poor performance compared to SOTA methods that use search and simulations. For future work, we would like to explore planning methods and different exploration strategies (e.g. model-predictive control, Thompson sampling) to improve upon the short-sighted value estimation. Instead of adding more auxiliary tasks, we would also explore different intermediate supervision strategies such as Hindsight Experience Replay (HER).

Improving Q-Learning Sample Efficiency with Representation Learning for 2048

Rachael Cooper
Department of Computer Science
Stanford University
rcooper6@stanford.edu

Melinda Zhu
Department of Computer Science
Stanford University
melinda7@stanford.edu

Abstract

This project aims to combine several Q-learning and representation learning-based approaches to develop an agent capable of achieving high scores while minimizing the number of training episodes required to do so. We first implement a Deep Q-Network baseline trained on flattened vector representations of the game board, and then we introduce spatial awareness and exploration guidance through a CNN-based encoder, auxiliary tasks, and reward shaping. The enhanced DQN consistently outperformed the baseline, achieving a higher maximum tile and demonstrating significantly fewer illegal moves within the same number of training episodes. These results highlight the importance of structured state representations and auxiliary learning signals for sparse-reward tasks.

1 Introduction

Gabriele Cirulli's 2048 is a popular puzzle game in which players aim to merge tiles on a 4×4 grid to reach the highest possible 2^n-numbered tile until no more merges can be made. A new 2- or 4-value tile is spawned at a random empty location after every move, making the game unpredictable and difficult to build strategy around. The reward structure is sparse – most moves yield no reward, and many high-reward sequences require priming advantageous tile positions to minimize low-value merges and overcrowding. Because of the game's non-deterministic nature, simply making random moves tend to lead to a swift game-over. Without deliberate planning or learned strategies, agents either merge small tiles inefficiently or make short-sighted decisions. Throughout several trials, the agent should ideally be able to veer away from naive, greedy approaches that maximize immediate score or randomly spamming illegal moves (i.e. moves that result in no slides or merges).

To address this, our project aims to improve sample efficiency – the agent's ability to learn effective policies from limited experience – by leveraging deep reinforcement learning approaches. This is a challenging problem to solve, as current popular approaches such as search methods or handcrafted heuristics typically rely on brute-force simulations of possible move sequences, which require millions of runs and high-end hardware [2]. This problem is widely applicable to general decision-making tasks such as resource allocation, robotics, or other sparse-reward games like Tetris, where exhaustive searches are expensive and do not contribute to extrapolating useful strategies long-term. In our implementation, we focus on experience replay and extrapolating strategies from tile positions and learned features from the board, without relying on hardcoded rules nor expensive simulations.

Figure 1: Sample 2048 board state, with max tile 32.

Our approaches are based upon a model-free Deep Q-Network, which has shown strong performance in discrete action environments like Atari games [7]. In addition, we incorporate a convolutional neural network (CNN) encoder to create feature representations of the board's spatial structure that can capture tile locality and merge opportunities. To further drive productive learning, we introduce auxiliary tasks that establish intermediate goals and help to generalize strategies. Lastly, we apply reward shaping to give bonuses on productive decisions as well as guide the agent away from illegal moves while still encouraging exploration. To evaluate the impacts of each experiment, we analyze the per-task loss over time, max tile reached and number of illegal moves per episode, and total rewards throughout training. Qualitatively, we calculate the Gini coefficient for each episode to determine the strategy of the agent and how it moves tiles around the board, as well as recorded the max tile achieved.

Through these approaches, we demonstrate that our agent not only surpasses the baseline DQN in terms of maximum tile reached and making more efficient, legal moves. However, the agent fails to surpass a maximum tile ceiling of 512 in 10,000 steps and falls significantly below the bar of current 2048 reinforcement learning models. Even so, this work establishes a proof-of-concept that highlights how structured representations and auxiliary supervision can enhance the learning process.

2 Related Work

Several previous works have explored solving the 2048 game with reinforcement learning. In 2022, Guei achieved SOTA performance for 2048 using optimistic temporal difference learning, but left deep RL mainly unexplored, which we intend to focus on [2]. With the goal of producing a proof-of-concept of achieving decent performance from DRL without high computational demand for 2048, Guei tested a combination of Gumbel MuZero and Stochastic MuZero (variants of algorithms built on top of MCTS with a dynamics model to predict the next state/reward). This experiment achieved a score of 394,645 with just three training simulations, demonstrating the feasibility of deep RL approaches for 2048. However, in this work, there was no end-to-end deep RL framework or any representation learning implemented; raw board states were used without any abstract features of the layout (e.g. tile clusters or symmetry, which can help to generalize across strategies).

The exploration of deep RL for 2048, carried out by Kaundiya et al. with Q-learning and SARSA (non-representation learning methods), used the raw 2048 board as input and relied solely on tile values to learn the optimal reward [6]. The resulting agent learned promising behaviors (e.g. high values in the corners), but the agent was not able to surpass a score of 512. Kaundiya et al. attributes this result to overfitting and thus a lack of generalization, feature understanding, and randomness adaptability. In addition, the model had poor sample efficiency results. In our project, we intend to build upon this framework by incorporating representation learning to improve sample efficiency and surpass the 512 maximum tile.

Similarly, Sauren et al. attempted to use RL to solve the 2048 problem by including expert knowledge in the form of strategic constraints [9]. This interpretation is evaluated against a "safe" model with shielding, and a "risky" model. This method improved reliability, but had limitations such as training instability, rigid heuristics, and frequent manual intervention. Since the model heavily utilized hand-crafted rules, this led to a lack of self-adaptability and required constant guidance to perform well. In contrast, deep RL, which is what we intend to utilize instead, enables agents to learn both representations and policies more independently, removing the need for manually designed heuristics.

3 Method

3.1 Baseline Deep Q-Network

For our baseline reinforcement learning method, we implemented a vanilla DQN [7]. In 2048, each state of the task is represented by a flattened vector of the board, where each entry is a tile's value (or blank if it is empty). To reduce numerical instability, we transformed the raw tile values into $\log 2$. Our DQN consists of a neural network with two hidden layers and ReLU activations, mapping the flattened vector to four Q-values corresponding to the four possible actions (up, down, left, or right). During training, the agent selects actions according to an ε-greedy policy; we chose this policy since it is an effective approach that balances exploration and exploitation during training, and encourages

the agent to explore the state-action space early on. We also introduced decay to ε throughout training to evolve the agent into making more value-based decisions.

We chose this shallow Q-network architecture because the 2048 board has a low-dimensional state space (4×4), and deeper networks may fail to capture the simple rules and spatial patterns on the small grid. Two hidden layers are typically used in baseline DQN implementations, and ReLU activations are used primarily to mitigate vanishing gradients (which also become more of a risk for deeper networks with more high-dimensional layers) [3]. Moreover, since our project is focused on *sample efficiency,* we preferred experimenting with simpler networks that require less compute for training.

The agent stores transitions (s, a, r, s', d) in a replay buffer, where the agent took action a in state s, received reward r, and landed in the next state s', which d indicating whether the game ended in that episode. During training, the agent samples random batches of these transitions to update the Q-network. We used this experience replay method to primarily improve data efficiency (as reusing past experiences reduces the need to simulate more interactions with the environment). For the loss function, we use MSE since the network is trained to estimate the expected cumulative reward for taking an action in a given state.

3.2 Convolutional Neural Network Encoder

Since the vanilla DQN takes in a flattened vector as input, it fails to capture tile arrangements, which the agent must be able to understand to make smart decisions. Thus, we introduced a CNN encoder in order to capture local patterns and spatial dependencies. For example, merging high-value tiles in the corners of the board or maintaining more empty space are not so easy for an agent to realize if their experience is based on a flattened vector. The goal of the CNN in terms of improving sample efficiency is that the agent can learn strategies faster based on a rich understanding of tile positioning instead of doing brute-force trial and error.

The CNN network processes the 4×4 grid directly; the architecture consists of a set of 2D convolutional layers that extract spatial features at varying resolutions, followed by FC layers that map these spatial features to Q-values. More specifically, the first convolutional layer applies a 2×2 kernel to capture more fine-grained local patterns (e.g. adjacent tiles that can merge together). The second convolutional layer builds on these features by further expanding the receptive field to capture broader structural features with another 2×2 kernel, expanding on the previous layer and increasing the number of channels, resulting in a $2 \times 2 \times 64$ feature map. This is then flattened into a 256-d vector and passed through a FC layer similarly structured to the vanilla DQN for consistency. This output embedding can be used for auxiliary tasks and the main Q-value predictor (discussed further in 3.2.1 and 3.2.2). Once again, these architectural choices were made based on optimizing for efficiency, since CNNs are more efficient compared to dense NNs when the input data is structured as a grid.

We also incorporated 2D batch normalization and dropout: canonical strategies for achieving better loss convergence and training stability for CNNs [4] [10]. These choices were useful for 2048 since signals are noisy and sparse, and normalization aids with preventing overfitting and encouraging generalization across states that share key spatial features.

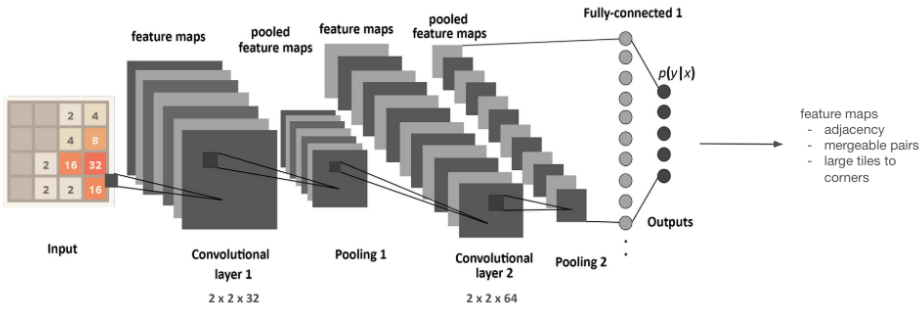


Figure 2: CNN architecture for creating feature maps from a 2048 board. The network consists of two convolutional layers using `kernel_size=2` (with 32 output channels and 64 output channels, respectively) with max pooling.

3.3 Auxiliary Tasks and Intermediate Signals

Given the sparsity of 2048, we incorporated self-supervised auxiliary tasks that help provide intermediate signals to the agent to learn advantageous strategies. Built on top of the CNN encoder, the auxiliary heads further enrich the state representations, as they push the network to learn features that are not just useful for maximizing reward, but also for predicting other properties of the environment. For the implementation, each auxiliary tasks adds a lightweight MLP (`Linear` $\rightarrow$ `ReLU` $\rightarrow$ `Linear` architecture with a loss function that matches up with the task type). Since the output is a regression (estimating $\log_2$ of the highest tile), we used an MSE loss.

3.3.1 Auxiliary Task: Action Legality Prediction

The first auxiliary head trains the agent to predict which actions are legal given the current board state. In 2048, the actions that correspond to an illegal move (i.e. no merges or tile movement) vary depending on the board's tile configuration. By training a legal move classifier, the agent is able to receive an intermediate signal that helps it learn constraints as well as tile positioning to optimize for advantageous merges, as opposed to trying random actions. This task improves both exploration and value estimation by reducing the amount of updates based on illegal transitions which are added to the replay buffer. Through this structured exploration guidance, the agent can learn features that build a more accurate model of the action space. Since there is only one output per action for this auxiliary task, we used binary cross-entropy loss for the MLP.

3.3.2 Auxiliary Task: Maximum Tile Prediction

The second auxiliary task trains the agent to predict the maximum tile that will eventually be reached on the board after a given move. This task aims to provide the agent with a more long-term signal. By being able to predict the maximum tile, the agent can gain a better understanding of what tile configurations and action sequences will lead to overall game progress, as in 2048, the main success indicator is what maximum-value tile a player is able to reach. This is distinct from learning immediate goals such as avoiding illegal moves.

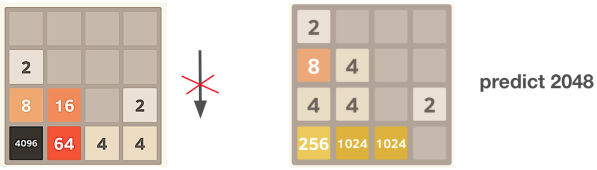


Figure 3: (left) Predicting whether the next action will be legal based on current tile positions. (right) Predicting what the max-value tile will be given the current tiles and potential actions to take.

3.4 Reward Shaping

In our setup, the reward is equal to the sum of tile values merged in each move. However, as the game progresses, the reward alone does not capture all "good moves" (e.g. setting up a future maximum tile merge or maintaining an advantageous layout yields zero reward). In other words, relying on the original reward as the only signal is insufficient for building up useful strategic representations that the agent can build around. To mitigate this, we applied reward shaping to guide the agent towards long-term advantages and better familiarity with board layouts.

First, we implemented reward shaping for **minimizing the number of nonzero tiles** (i.e. compression) after each move, which encourages board compactness as it allows players to make space for future high-value merges. To the reward, we added a small compression bonus based on the difference between the number of nonzero tiles before and after each move. This technique is one proxy for long-term success by helping the agent avoid situations where the board is too tense with small tiles.

$$\text{compression\_bonus} = \lambda_{\text{compression}}(\text{nonzero}_{\text{prev}} - \text{nonzero}_{\text{next}}) \tag{1}$$

Next, we gave a penalty for illegal actions after observing that the baseline DQN had high and increasing numbers of illegal moves. While illegal moves may seem like they do not have any impact

on the game (i.e. no board movement doesn't end the game or make it more difficult to reach a high tile), they indicate poor board awareness or taking several random unjustified actions. Through a penalty (also configured as a hyperparameter), the policy is encouraged to reduce unproductive behavior. We decided to implement this instead of explicitly blocking all illegal moves, as it still allows **exploration** and the agent to learn from its mistakes while gaining a better fundamental understanding of how merges work and how to optimize its actions.

4 Experimental Setup

The model and agent code were built using PyTorch, and was made modular to allow for customizability (e.g. turning off the CNN encoder, or providing different task weights for the auxiliary heads). We trained and evaluated our custom DQN agent via several different evaluation techniques. Initially, we began with hyperparameter tuning, and then conducted an ablation study to isolate the effects of each implementation as well as compared the results to the baseline DQN. Hyperparameter choices such as the learning rate, batch size, and target update frequency were tuned empirically based on training stability and observation of progress. Finally, we did a qualitative evaluation of the game strategy through Gini coefficients and max tile achieved. To track training progress without waiting for entire runs to complete, we added a Tensorboard writer to record metrics.

The custom DQN agent was trained on `gymnasium_2048` [1], wrapped in a custom class to cast the per-step scores to floats due to several issues with overflowing values. Board values were also normalized with $\log_2$ to mitigate memory errors and stabilize training. The environment wrapper class provides the following:

- Action Space: {0: up, 1: right, 2: down, 3: left}
- Observation Space: one-hot tensor ($4 \times 4 \times 16$)

All experiments used a standard 4×4 board and were seeded equally (`seed = 42`). In addition, all experiments are trained on a NVIDIA RTX 4080 (16 GB), Intel Core i9 CPU, and 128 GB RAM. To evaluate sample efficiency and conduct a sufficient number of experiments, we trained each experiment for 10,000 episodes except for the hyperparameter search in which we trained many iterations for 300.

5 Results

The quantitative results we gathered for our agent show measurable outcomes, such as total rewards, convergence speed, and improvements from hyperparameter tuning. The qualitative results focus on interpreting the agent's behavior and Gini coefficient to provide deeper insights beyond raw performance, as well as the max tile achieved.

5.1 Quantitative Evaluation

5.1.1 Hyperparameter Search

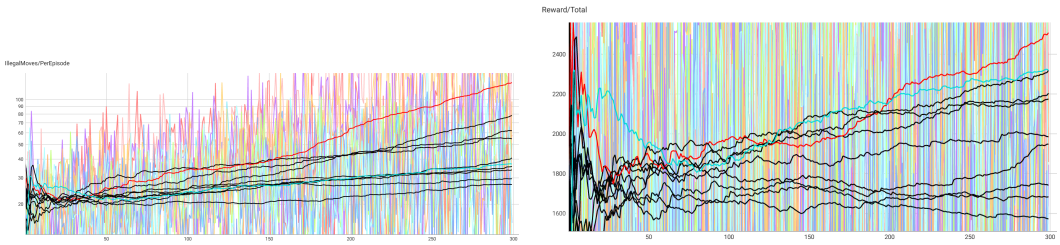
To initially optimize the performance of our DQN model, we conducted an extensive hyperparameter search on the following bounds found in Table 1.

Table 1: Hyperparameter Search Bounds

Learning Rate	Gamma	Epsilon Decay	Illegal Move Penalty	Reward Shaping Max Tile Weight	Compression Shaping Weight	Legal Aux Weight	Max Tile Aux Weight
1e-5 - 1e-3	0.90 - 0.99	10000 - 30000	0 - 2	0 - 2	0 - 2	0 - 0.5	0 - 0.5

To compute this hyperparameter search, we used the Optuna optimization library to enable efficient, automated tuning via a form of Bayesian optimization, so we could explore a wide range of configurations with minimal manual intervention. Each configuration was tested over multiple runs to

account for any randomness in the training. These runs were computed each for 300 steps to gain an initial understanding. Each trial used a unique combination of the hyperparameters in the ranges listed above.



(a) Illegal Moves (b) Reward

Figure 4: Hyperparameter Training

Figure 3 depicts the best trials run on both illegal move performance and reward. In the graphs above, although the red line had the highest reward, the number of illegal moves far surpased the others, which demonstrated a lack of true learning. On the otherhand, the blue line demonstrates a high reward along with one of the lower illegal move counts. We determined the blue run to be our best run. The hyperparameters used are listed in Table 2.

Table 2: Best Hyperparameter Search Result

Trial	Learning Rate	Gamma	Epsilon Decay	Illegal Move Penalty	Reward Shaping Max Tile Weight	Compression Shaping Weight	Legal Aux Weight	Max Tile Aux Weight
9	1e-4	0.98	23073	0.85	0.74	0.28	0.17	0.20

We used these hyperparameters as the baseline for all further analysis.

5.1.2 Ablation Study

To better understand the contribution of the auxiliary rewards we added to the DQN agent, we conducted an ablation study by systematically reintroducing each feature to an otherwise minimal configuration. We set each variable to be "on" at a time, and the others as "off" (set to zero) to see how each individual component affected the success of the model. All experiments were run under the same environment settings for fairness. We ran each iteration across 10,000 steps and compared each with the baseline run using the hyperparameters found in the hyperparameter search.

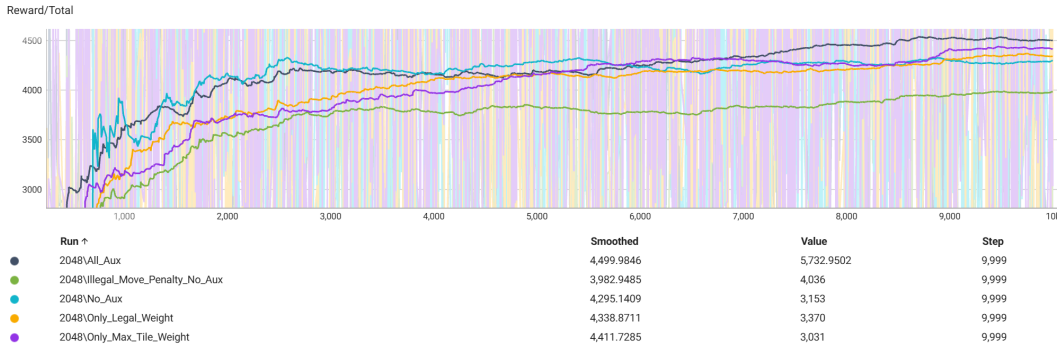


Figure 5: Total Reward per 10000 Steps

The baseline distinctly outperformed the other iterations in the total reward as seen in Figure 4; however, for the number of illegal moves in Figure 5, they all performed generally the same. The

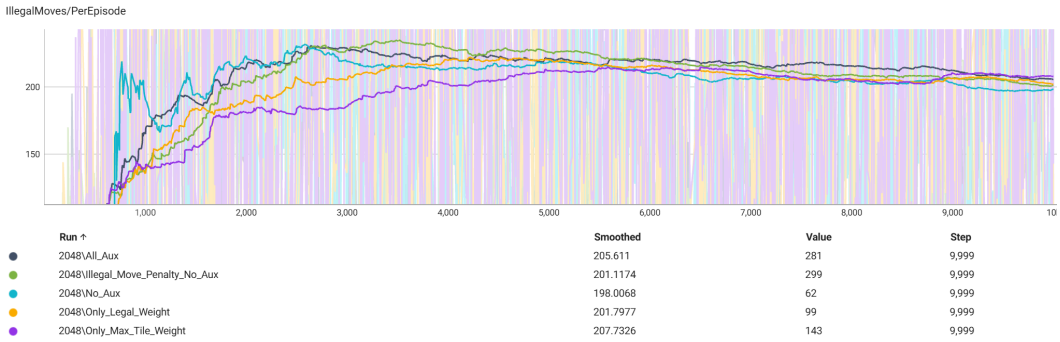


Figure 6: Illegal Moves per 10000 Steps

illegal moves hyperparameter configuration performed the second best, but with more steps the results may demonstrate an eventual decrease below the others.

The mechanism that contributed to the lowest total reward, was the illegal move penalty, but proved necessary on the illegal move count graph. The second highest reward was achieved by the maximum tile prediction auxiliary task weight, which correlates with our understanding as obtaining higher tiles in gameplay provides the biggest increase in the overall reward.

5.1.3 Baseline Comparison

To evaluate the effectiveness of the proposed improved DQN agent, we compare our model with a baseline DQN implementation. It is important to note that the baseline DQN approach uses a separate reward function that has negative values, whereas we customized ours. As a result, while the direct reward comparison may not be entirely fair, the comparison between the number of illegal moves remains useful for analyzing behavior such as learning stability and policy robustness. Both models were run for 10,000 steps each and the results are shown in Figure 7.

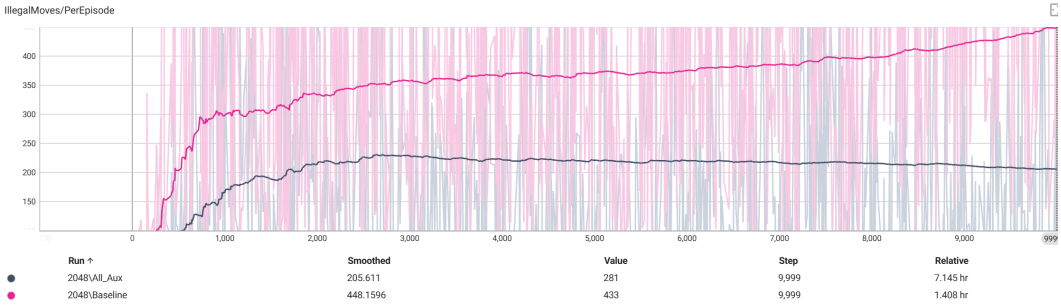


Figure 7: Illegal Moves Baseline Comparison

Our agent outperformed the baseline DQN by reducing the number of illegal moves by around 300 over the course of 10,000 steps. This significant difference demonstrates that the baseline model failed to properly learn a valid policy for playing the 2048 game. On the otherhand, our agent demonstrates a stronger understanding of the strategy, most likely do to the additional mechanisms like the CNN and auxiliary tasks.

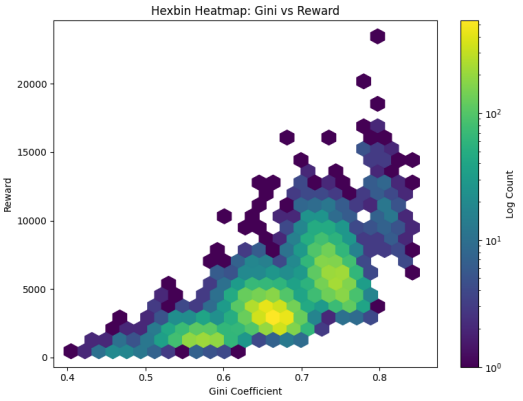
5.2 Qualitative Analysis

5.2.1 Gini Coefficient & Agent Behavior Observations

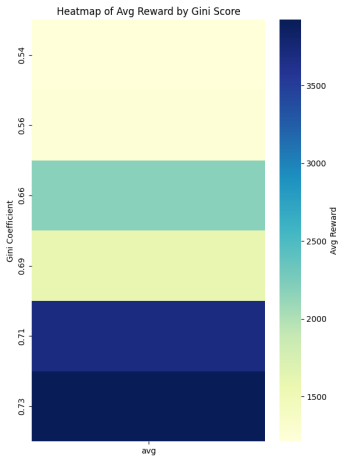
The Gini coefficient measures inequality. Specifically, in the context of the game 2048, the Gini coefficient tracks tile distribution. It measures how unevenly the tiles are spread across the game board and is a strong qualitative indicator of whether the agent has learned such a strategy. The coefficient is calculated by the formula:

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n^2 \bar{x}}$$

where n is the number of tiles on the board, x_i and x_j are the individual tile values, and $\bar{x}$ is the average of the values. If the value of G is 0, that demonstrates perfect equality (there are many tiles and they all have the same value). If the value of G is 1, there is perfect inequality (one tile has all the value). In a game of 2048, a higher Gini coefficient is more desirable as the agent would be successful in merging small tiles into larger ones [8].



(a) Gini Coefficients in Relation to Total Reward



(b) Heatmap of Gini Coefficient Compared to Total Reward

Figure 8: Visualization of Gini coefficients and their relation to total reward.

The graphs demonstrate a clear upwards trend where as the Gini coefficient increases, so does the reward. This corroborates our understanding that when the board has more unequal tile distributions, the model tends to achieve higher reward. When run on our custom DQN, we can see three high density areas around the 0.65-0.70 range, meaning most episodes tend to have moderate merges and improvements can still be made to the strategy of the agent. When the agent does achieve a high Gini score, the reward is significantly higher.

5.2.2 Max Tile

Additionally, the maximum tile achieved for both the baseline and our custom DQN can be seen in Table 3.

Table 3: Max Tile Achieved in 10,000 Episodes

Baseline	**Custom DQN**
256	512

Our custom model was able to outperform the vanilla DQN, but a maximum tile of 512 is still underperforming.

6 Discussion

Adding representation learning techniques to the DQN setup produced agent behaviors indicative of more calculated moves and a better understanding of the game rules. With the baseline DQN, the agent made **random, short-sighted** moves without any progress in learning what a legal move was, and never surpassed the maximum-value tile achieved near the start of training. On the other hand, the CNN with auxiliary tasks and reward shaping drastically reduced illegal actions, and the agent was

able to progress further in the game **with fewer episodes**, thus increasing sample efficiency compared to the baseline. The auxiliary tasks that we introduced (legal action prediction and maximum tile prediction) provided vital intermediate learning signals that guided the agent beyond what the sparse reward alone could signal. In other words, the agent was able to learn a deeper representation of the game's constraints and more optimal moves for long-term success in the same amount of training time. Reward shaping also helped the agent steer towards general strategies; i.e. "less nonzero tiles on the board can make more space for high-value merges."

However, our primary takeaway from this work is that the agent, despite incorporating strategy-based representation learning, still could not get past the very low maximum tile value of 512 in solely 10,000 episodes. This plateauing behavior highlights the challenge of self-learning long-term strategies without being explicitly told what to do. Without planning/search methods, the agent still struggles to discover beneficial strategies (e.g. preserving corner tiles or monotonic sequences) which are essential to scaling up larger tiles [6]. Although the auxiliary tasks and reward shaping helped to nudge the agent towards pursuing more strategic behaviors, it failed to realize the benefits of such strategies. For example, adding the maximum tile prediction task did not improve over the other techniques even though it was specifically geared towards helping the agent choose actions that would increase the current maximum-value tile on the board.

We attribute this result to how the agent's value estimates are still based on short-term targets in Q-learning for sparse reward tasks [5]. A key strategy in 2048 is being able to organize low-value tiles in specific ways to achieve higher-value merges later on. With the DQN and representation learning methods we employed in this project, the lack of planning or lookahead searches causes the agent to primarily focus on immediate value which is also why episodes are short and there was an evident maximum-value tile ceiling. Another example is that even though the agent can get more rewards from minimizing nonzero tiles (one of the reward shaping additions), it still fails to understand the overall long-term benefit of doing so, and still finds itself reaching game-over due to the game board becoming overcrowded quickly.

In addition, since the agent's exploration strategy is limited to ε-greedy, it cannot adapt well as the board states evolve. This makes the exploration relatively random and does not allow for the agent to effectively utilize the learned representations and their overall meaning in the context of trying to achieve long-term success. Furthermore, it is also highly likely that agents need significantly more training episodes to breach higher max-value tiles and realize effective strategies [2]. In our setup, we were limited by memory management, training stability, time, and computational power constraints. However, longer training alone likely will not solve the plateauing problem – without better exploration and planning incorporated into the architecture, the agent may still converge to local minima and short-sightedness.

7 Conclusion & Future Work

Our work examined the use of reinforcement learning, specifically DQNs, to play the game 2048. We used a vanilla DQN as a basis for our experimentation, and then incorporated a CNN encoder, reward shaping, and auxiliary tasks to help the agent learn policies that were more spatially-aware. Specifically, these additions reduced the number of illegal moves as well as improved the model's navigation of the 2048 environment. In comparison to the baseline DQN, our custom model exhibited more promising strategies, such as reaching higher tile values. This suggests a recognition of the successful strategy and gameplay through structured learning signals.

Despite these improvements, our agent had a difficult time surpassing moderate tile values, plateauing early. This stagnant behavior demonstrates a fallback of the Q-learning approach, where a DQN's reliance on short-term reward estimation makes it ill-suited for mastering long-horizon strategies that are crucial in playing 2048. Thus, for our future work, we plan to explore planning methods and more adaptive training strategies. In particular, integrating model-based rollouts, lookahead planning, or Hindsight Experience Replay (HER) could help the agent plan and learn strategies over longer horizons and also learn from "failed" episodes: e.g. premature endings, too many insufficient merges or overcrowded boards. We also aim to investigate more structured exploration mechanisms to reduce reliance on random ε-greedy behavior. Ideally, we can hopefully observe improvement within the same amount of training episodes as the experiments we conducted here to further test the effects of these other additions on sample efficiency.

8 Team Contributions

- **Equal Contribution:** poster and one-page abstract
- **Rachael**
  - **Implementation:** Implemented the 2048 environment, the evaluation metrics, and experimentation (hyperparameter search, ablation study, Gini coefficients).
  - **Report:** Prior works, experimental setup, results, and conclusion sections.
- **Melinda**
  - **Implementation:** Implemented the baseline DQN, CNN encoder, agent/network modules, training loop, auxiliary tasks, and reward shaping.
  - **Report:** Introduction, methods, and discussion sections.

Changes from Proposal Minimizing illegal moves was not planned from our proposal but ended up being a critical metric that tracked the agent's awareness of how to play the game properly based on tile layouts. Reward shaping was also not part of our original plan from the proposal, but we decided to implement it in order to incorporate more explicit signals that could aid the auxiliary tasks and CNN encoder.

9 Project Code

https://github.com/melindazhu/CS224R-2048-Representation-Learning

References

[1] Quentin Delfosse. gymnasium-2048: Reinforcement learning environment for 2048. `https://github.com/Quentin18/gymnasium-2048`, 2024. GitHub repository, accessed: 2025-06-08.

[2] H. Guei. On reinforcement learning for the game of 2048, 2022. Ph.D. dissertation, Institute of Computer Science and Engineering, National Yang Ming Chiao Tung University. Advised by Dr. I-Chen Wu.

[3] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[5] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[6] V. Kaundinya, S. Jain, S. Saligram, C. K. Vanamala, and B. Avinash. Game playing agent for 2048 using deep reinforcement learning. In A. K. Jha, editor, *Proceedings of International Conference on Computing, Communication, Electrical and Electronics Engineering*, pages 369–373. AIJR Publisher, 2018.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[8] Max Roser and Esteban Ortiz-Ospina. What is the gini coefficient? `https://ourworldindata.org/what-is-the-gini-coefficient`, 2013. Accessed: 2025-06-08.

[9] K. Sauren. Improving reinforcement learning performance in 2048 using expert knowledge, 2022. Bachelor's thesis, Radboud University. Advised by Dr. Nils Jansen and assessed secondarily by Dr. Daniel Strüber.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

A Additional Experiments

A.1 Baseline Comparisons

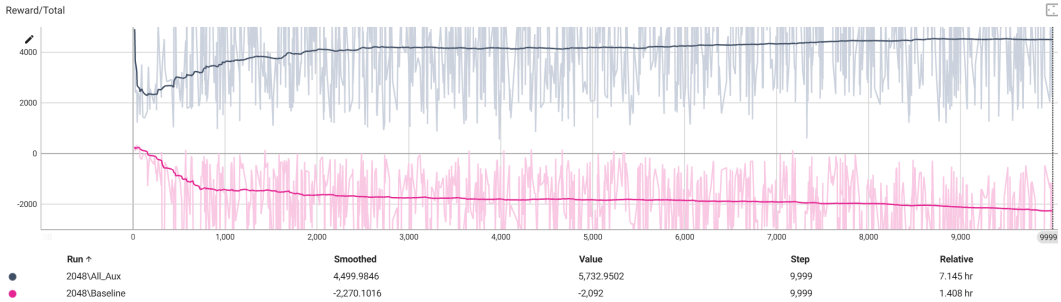


Figure 9: Total Reward per 10,000 Episodes

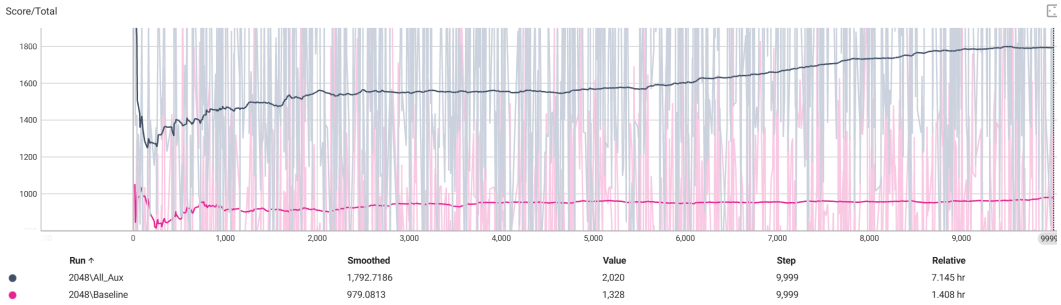


Figure 10: Total Score per 10,000 Episodes

A.2 Auxiliary Task Loss

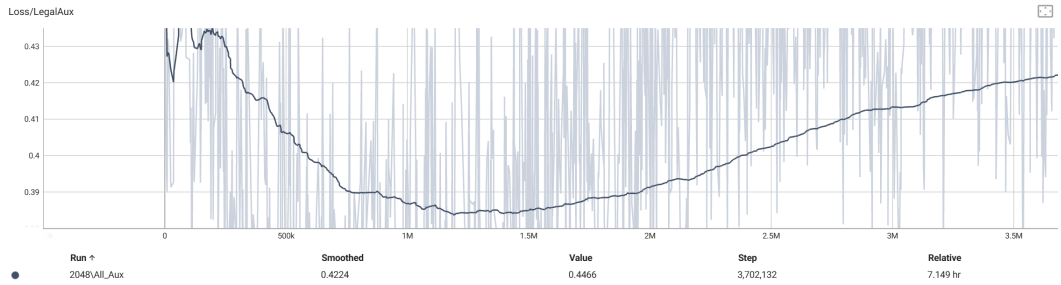


Figure 11: Auxiliary Task: Legal Moves Loss per 10,000 Episodes

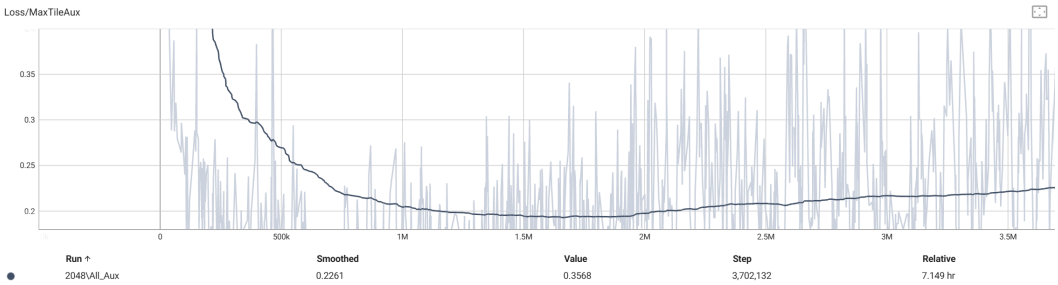


Figure 12: Auxiliary Task: Max Tile Loss per 10,000 Episodes

A.3 Reward Bonuses

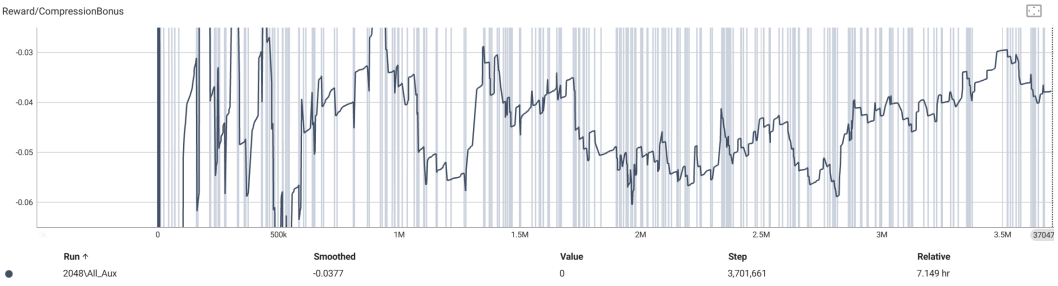


Figure 13: Compression Reward Bonus per 10,000 Episodes

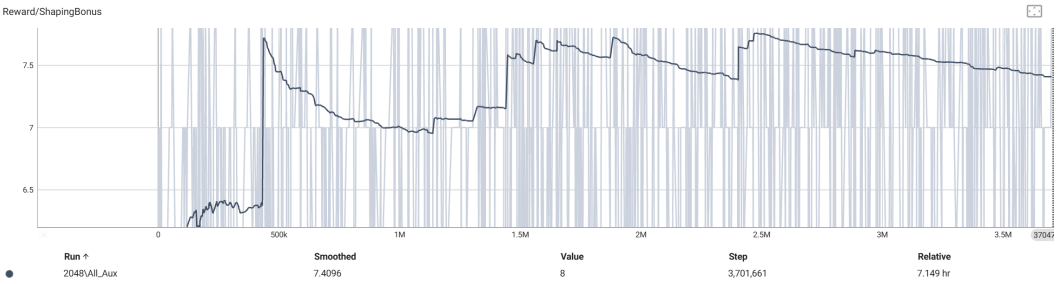


Figure 14: Shaping Reward Bonus per 10,000 Episodes