Reinforcement Learning Final Report - Learning how to play 2048 using TD(λ) learning over n-tuple networks

Maadhav Gupta

April 30, 2025

1 Introduction

The game has simple rules, a stochastic environment, and long-term strategic depth. The objective is to combine tiles by sliding them in one of four directions to reach the 2048 tile, while maximizing the score. Despite the deterministic mechanics, the insertion of new tiles introduces randomness that gives the environment a bit of randomness.

This project explores learning an effective 2048-playing agent using temporal-difference learning with eligibility traces, combined with n-tuple networks for value function approximation. The goal is to learn from raw interaction with the game without access to search-based planning or tree expansion, relying purely on model-free learning. A key challenge in this task is the high-dimensional and partially observable state space, which demands efficient and generalizable state representations.

2 Literature Review

Goenawan, Tao, Wu (2020) attempted an MCTS (Monte Carlo Tree Search) for the game. They considered the total sum of the tiles on the final board, the total 'merge' score (say, when 2 and 2 get combined to become 4, that 4 is added to the score) and the value of the biggest tile on the board for the value function. They discovered that the sum of the tiles on the final board ends up delivering the best results. On the other hand, when considering the value of the biggest tile, it leads to the worst results of the lot.

Nathaniel Thomas has a website for the game with an interactive UI where you can visually see the algorithms that he's used in play. One of the algorithms is the Monte Carlo Tree Search algorithm to choose the best move. It simulates many games from the current state, and chooses the move that leads to the highest average sum of tiles. It is claimed that it wins around 75% of the time. He also uses n-tuple Network trained using Temporal Difference Learning for over 3M+ games, which is said to should win almost 100% of the time, and it gets the 16384 tile around 60% of the time. Lastly, he also utilizes an Expectimax algorithm to search through all possible game states and choose the move with the highest likely score. This is claimed to achieve the 32768 tile around 30% of the time, however it is much slower than the other algorithms.

The concept of n-tuple networks was originally introduced by Lucas (2008) in the context of Othello and extended by later works to Connect-4 and 2048. An n-tuple network consists of multiple fixed tuples of board positions, each mapped to a lookup table of values representing estimated returns. The value of a board is then approximated as the sum of the values across all tuples. This approach provides a nonlinear, distributed form of generalization that is simple to implement and fast to evaluate. In Connect-4, the authors used temporal-difference learning with n-tuple networks and demonstrated that a well-designed set of tuples (e.g., rows, columns, diagonals) could effectively capture local board structures and strategic interactions. Their success in a two-player adversarial game suggests that n-tuple networks are capable of learning meaningful patterns even in complex strategic environments.

3 Datasets source and Description

The data is generated through self-play using the game environment, where the agent interacts and learns via rewards.

I have used this code to develop the logic of the game. The action set involved and the state spaces are simple to define. The code that I have referred to does this in an elegant manner and allows experimenting.

4 Data Exploration and Analysis

Understanding how the agent performs across training episodes and different configurations is crucial to diagnosing learning behavior and improving algorithmic decisions. In this project, several key metrics were tracked during training to guide development and evaluation:

4.1 Score Distribution

Throughout training, the agent's game scores were logged periodically. This allowed for analysis of the distribution of scores over time. Early in training, the score distribution was narrow and centered around low values (200–400), indicative of mostly random or ineffective policies. As learning progressed, especially under the decaying ϵ-greedy policy, the distribution began to shift rightwards, with a noticeable increase in the frequency of scores above 800 and, eventually, above 1200. Histograms and moving averages of scores were used to visualize this trend.

4.2 Tile Achievement Frequencies

Another important metric tracked was the frequency with which the agent achieved high-value tiles such as 512, 1024, and 2048. These milestones serve as performance markers in the game. In the most effective configuration, the agent achieved:

- Tile 1024 in $\sim 50\%$ of the games,

- Tile 2048 in $\sim$ 3–5% of the games.

Tracking such discrete achievements is especially informative in 2048, as raw score alone doesn't fully capture the strategic depth required to build large tiles. The rolling standard deviation for the average score is increasing for the game as more episodes are observed, which is a feature of the game and not an anomaly.

4.3 Temporal Trends and Convergence

By plotting average game score over time (e.g., using a moving average over 1000 episodes), clear trends could be observed. The TD(0) approach showed early gains but plateaued, while TD(λ) with a decaying exploration rate exhibited a more gradual but continuous improvement. This suggested that a dynamic exploration strategy was more effective than a fixed one, especially in overcoming local optima.

4.4 Feature Impact and Tuple Efficacy

Experiments with different n-tuple configurations also revealed insights. Using only row-based tuples limited the agent's understanding of vertical and diagonal relationships on the board. Adding column and diagonal tuples significantly improved learning by providing a richer feature representation.

5 Methods

- TD(λ) with Eligibility Traces

  Temporal Difference learning with eligibility traces (TD(λ)) is used to update the value estimates for n-tuple features over sequences of states. At each time step t, we compute the TD error as:

  $$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

  For each active feature f in the current state s_t, the eligibility trace is incremented:

  $$e_t(f) = e_{t-1}(f) + 1$$

  Then all traces decay over time:

  $$e_t(f) \leftarrow \gamma\lambda e_t(f)$$

  The value of each feature is updated as:

  $$V(f) \leftarrow V(f) + \alpha\, \delta_t\, e_t(f)$$

  where:

- α is the learning rate,
- γ is the discount factor,
- λ is the trace decay parameter.

This approach assigns credit not just to the most recent feature activations, but also to features that were involved in earlier steps of the episode, proportionally to how recently they were active.

- n-tuple networks

  To capture useful spatial patterns, we use n-tuple networks. An n-tuple network is a hand-crafted feature extractor that selects fixed groups of tile positions (tuples) and learns a value estimate for each possible pattern observed in those tuples.

  In this implementation, multiple overlapping n-tuples are defined to extract information from rows and columns (e.g., 4-tuples of consecutive tiles). Each unique tuple pattern maps to an entry in a lookup table (value table), which stores an estimate of the value for that configuration.

- Value function approximation

  The value function $V(s)$, which estimates the expected future reward from state s, is approximated as the sum of the 'weighted average' of all active tuple features:

$$V(s) = \sum_{i=1}^{n} w_i[f_i(s)]$$

  where:

  - $f_i(s)$ is the index in the lookup table corresponding to the observed pattern in the i^{th} tuple,
  - w_i is the weight table for the i^{th} tuple.

  Each w_i is a preallocated array representing the value estimate for every possible configuration of the corresponding tuple. This structure allows the model to generalize across similar states by sharing weights among states that exhibit similar local patterns.

- ϵ-greedy with ϵ decay

  To balance exploration and exploitation, an ϵ-greedy policy is used during training. At each decision point, the agent selects an action a according to the following rule:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s,a) & \text{with probability } 1-\epsilon \end{cases}$$

  where:

- ϵ is a small positive number (e.g., 0.1), decaying over time to encourage exploitation in later training stages,
- $Q(s, a)$ is the estimated value of taking action a in state s, computed as the value of the successor state s' after applying action a.

In the context of the 2048 game, invalid actions (i.e., moves that do not change the board) are filtered out, and ϵ-greedy selection is applied only over valid actions.

6 Experimentation

Initial experimentation was conducted using the standard TD(0) approach. In this setup, a fixed learning rate $\alpha = 0.01$, fixed exploration rate $\epsilon = 0.1$, and discount factor $\gamma = 0.99$ were used. The feature representation employed only the four horizontal rows as n-tuples. While the learning showed some progress initially, performance plateaued after approximately 50,000 episodes, with the average score stagnating below 600. The program was run for a total of 6 hours, but no significant improvements were observed beyond this point.

Subsequently, a TD(λ) approach was attempted, with parameters set to $\alpha = 0.1$, $\epsilon = 0.1$, $\gamma = 1.0$, and $\lambda = 0.9$. A richer set of n-tuples was used in this case, including rows, columns, 2×2 blocks, snake-like patterns, and diagonals, as described earlier. However, the agent performed comparably to a random baseline, with average scores hovering around 400. Moreover, due to the increased complexity and overhead of maintaining eligibility traces with a larger number of features, the training process was significantly slower, taking around 18 hours. Overall, this configuration failed to show any substantial learning.

The current and more promising approach uses $\alpha = 0.01$, and a decaying ϵ-greedy policy where ϵ starts at 1.0 and decays by a factor of 0.9996 every episode, with a minimum threshold of 0.01. This ensures gradual reduction in exploration:

$$0.9996^n = 0.01 \Rightarrow n \approx 11{,}500 \text{ episodes}$$

This configuration, with $\gamma = 0.99$ and $\lambda = 0.5$, has shown significantly better learning behavior. The agent is now able to reach the 2048 tile in roughly 3–5% of games, and achieves the 1024 tile in about 50% of games. Performance continues to improve with ongoing training. The n-tuples used in this setup include rows, columns, and the two diagonals, striking a balance between representational power and computational efficiency. These results indicate that a combination of moderate λ values, decaying exploration, and a carefully selected set of n-tuples can lead to meaningful learning in the game of 2048 using temporal-difference methods.

7 Final Results

In Figures 1 and 2, we see that over time, the agent is able to learn how to survive better and achieve higher scores.

- Exploration is until episode ∼11,500.

- Once, it discovers a higher tile, it takes a while for the agent to consistently obtain the tile.
- The learning seems to plateau but I conjecture that once the agent is able to get more and more 2048's and possibly a 4096 and so on, it will be able to learn at a higher pace once again.

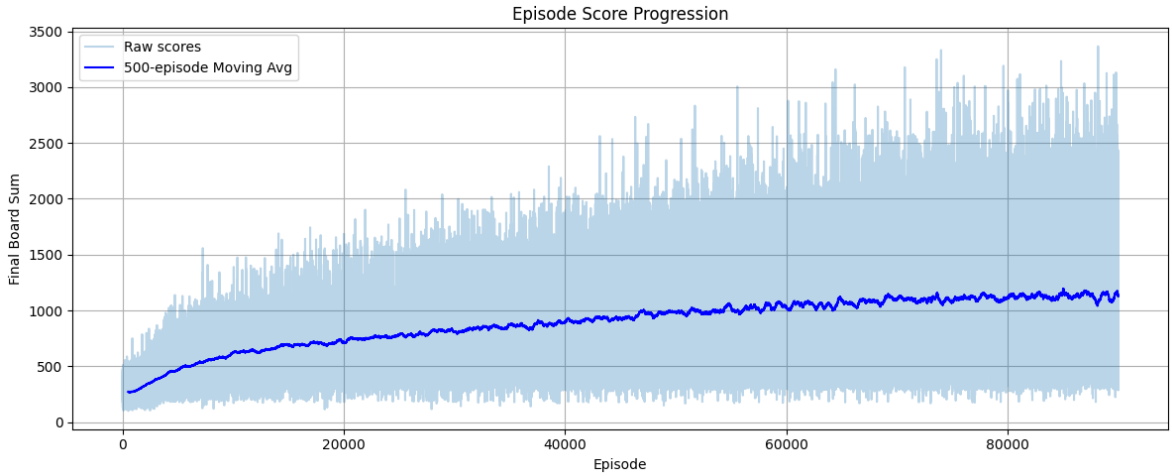

Figure 1: Average scores over episodes

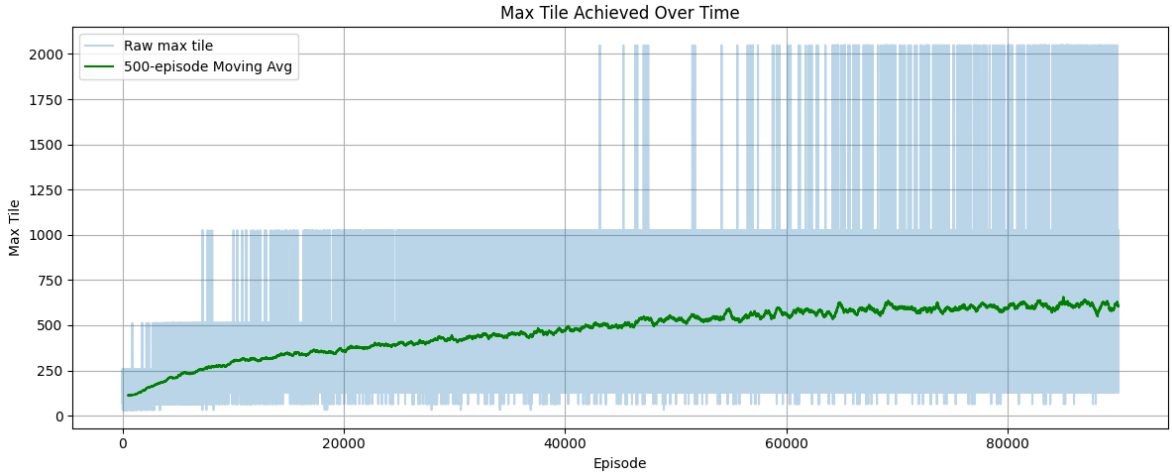

Figure 2: Max tile achieved over episodes

In Figure 3, we are able to see more clearly how the episodes change the frequencies of different max tiles. In Figure 4, the graph for frequencies of 1024 and 2048 tiles is more clear.

- Tile 256 which has the highest frequency very quickly loses its occurrence over episodes.
- Tile 512's occurrence is reducing but not as fast as the 512 tile.

- Tile 1024's frequency keeps on rising but the frequency in itself is quite varying. In some 100 runs it will show up a lot and in some it won't show up that much. This means the agent is still learning how to consistently achieve this tile.
- Tile 2048 is showing up quite late and the agent needs to train a lot more to learn to achieve this.

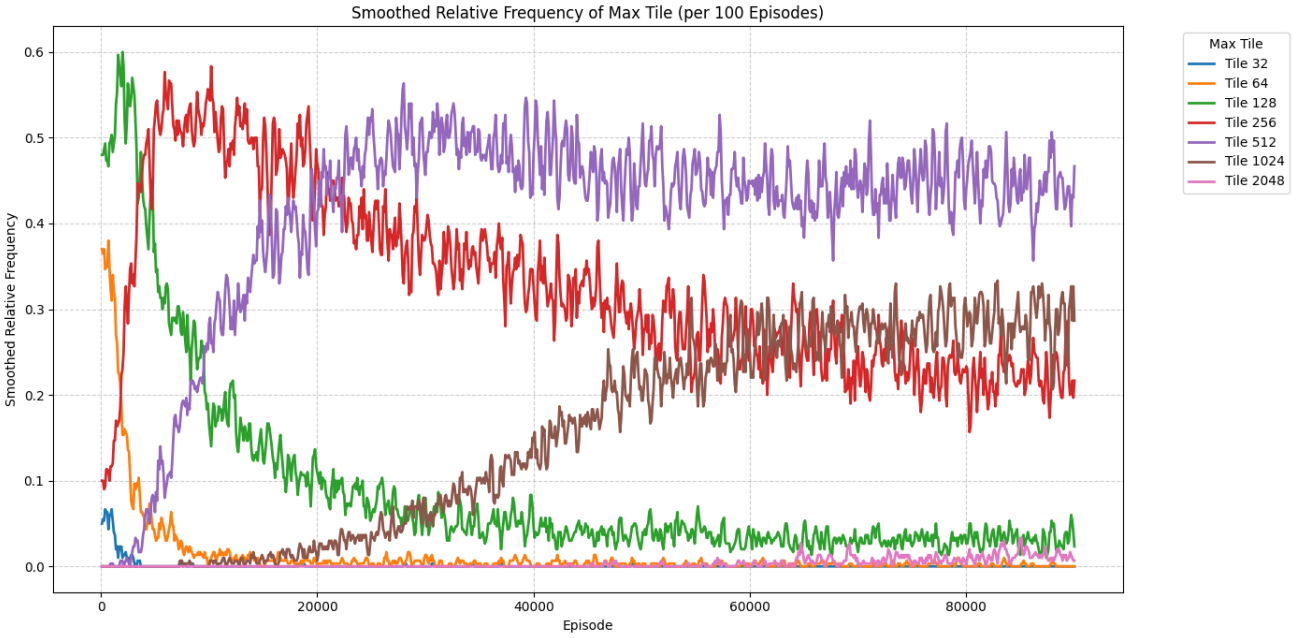

Figure 3: Smoothed relative frequency of max tile

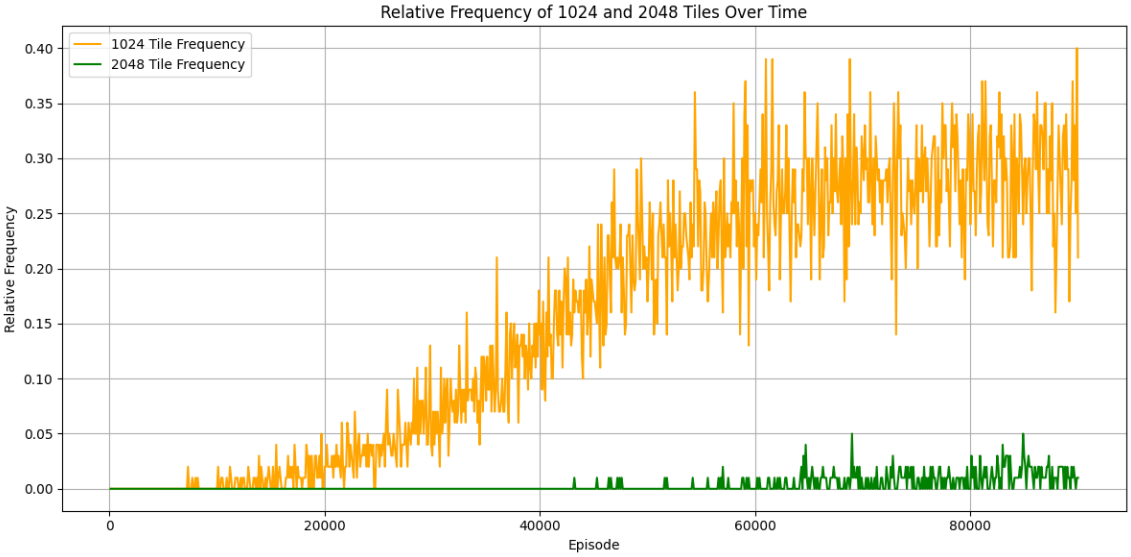

Figure 4: Relative frequency of max tile (for 1024 and 2048)

The standard deviations keep on increasing and once they start to stabilize is when the agent's learning will begin to saturate. There is some way to go for the agent to go.

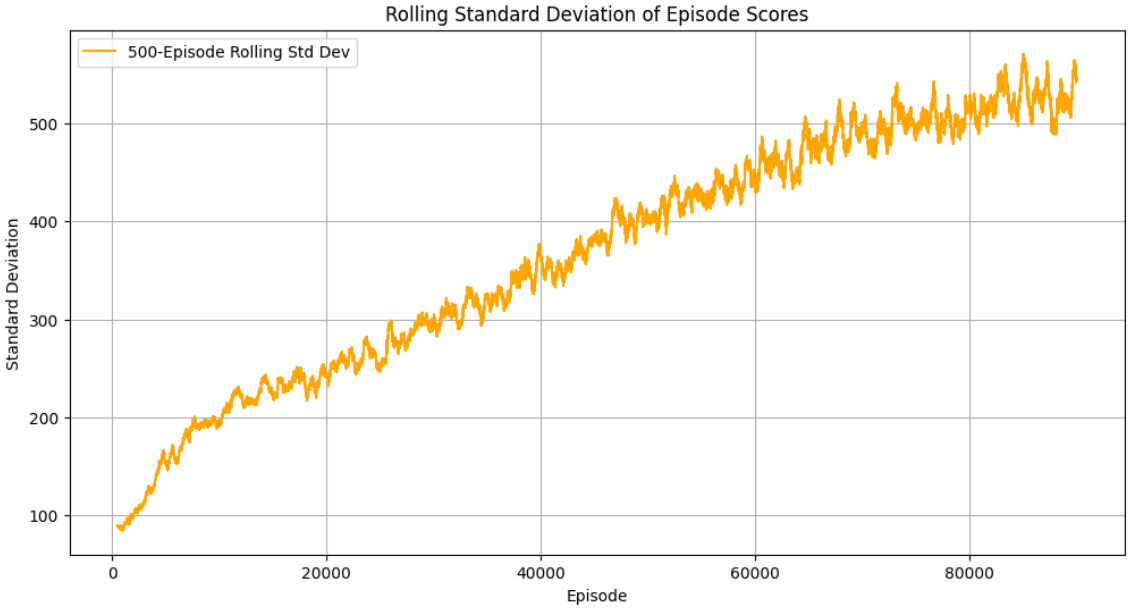

Figure 5: Rolling Standard Deviation of Episode Scores

The boxplot shows some promise as the 2048 tile to some extent dominates the 1024 tile and as more and more episodes go by, the Q-function should also realise the same and become better at what it does.

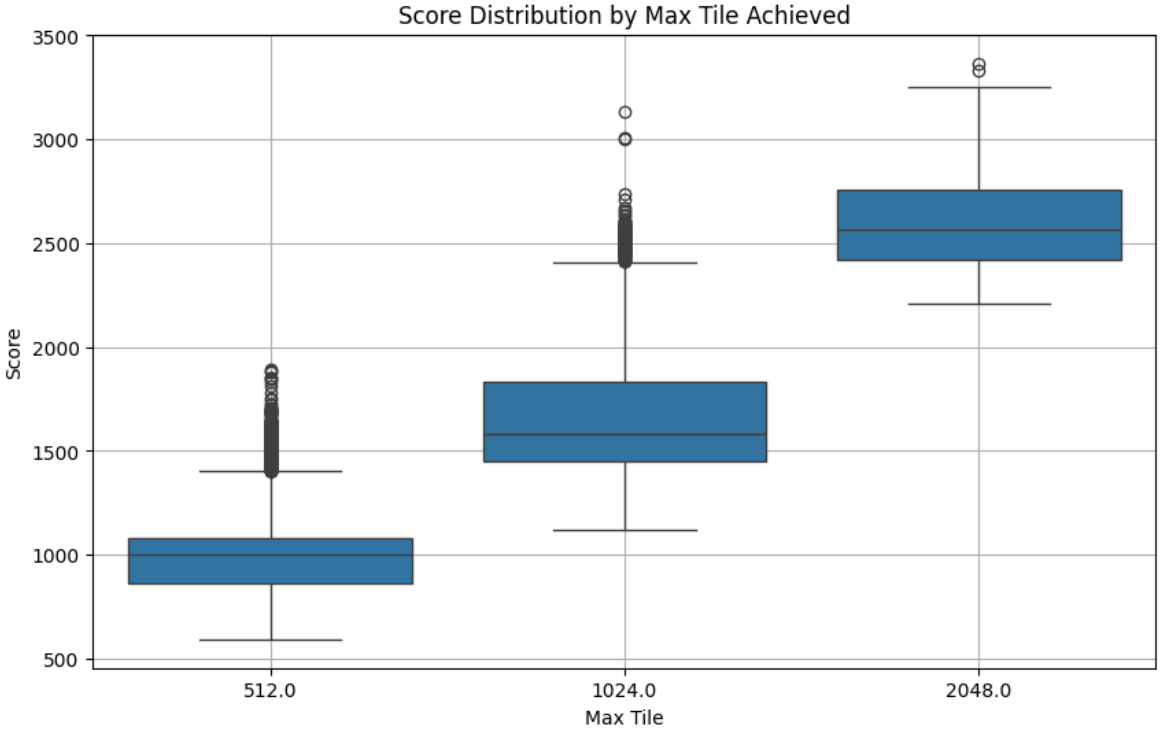

Figure 6: Boxplot for max tile achieved

8 Conclusion

The best agent is able to achieve the 2048 tile about 2.5% of the times after training for 92,500 episodes. Training it to 300,000 episodes and allowing more n-tuples as well as reintroducing exploration (by setting ϵ to, say, 0.5 every 15,000 episode episodes), agent should be able to improve in a better and effective and effective manner, at least theoretically.

We need the agent to get more consistent maximal tiles. For this, a challenge is for it to learn the basic heuristics of the game: monotonicity over rows, max tile number in a corner and high number of blank cells. One can only hope it can learn that on its own, and if not, it would be a certain improvement if some adjustments could ensure the heuristics are followed.

The agent's prospects seem hopeful but only time will tell what fate has in store.

9 References

- What's in a Game: Solving 2048 with Reinforcement Learning (Goenawan, Tao, Wu (2020))
- The Mathematics of 2048: Counting States with Combinatorics
- 2048.ai (Nathaniel Thomas (2024))
- poomstas
- n-Tuple Systems for Othello Learning (Lucas et. al)