**1912488642**

**Rifat Ibna Azad 1812298042**

**Play to Learn: Tic-Tac-Toe Using Deep Q-Network**

\*Note: Sub-titles are not captured in Xplore and should not be used

Name:Ruhsafa Haque

ID:1912488642

Name:Rafiqul Islam

ID:1821991042

Name:Rifat Ibna Azad

ID:1812298042

*Abstract*—This research investigates using reinforcement learning to train an AI agent to play the traditional board game Tic Tac Toe using a Deep Q-network (DQN). Instead of depending on preset rules or search-based algorithms, the goal is to create an intelligent agent to learn the best gameplay techniques by interacting with the environment. A 9-element state vector is used to represent the environment, which is described as a flattened 3x3 board. The Q-function is approximated by the agent using a neural network, which forecasts projected future rewards for every action that could be taken. The DQN model is trained using a mini-batch technique and experience replay, and it has two hidden layers with ReLU activations. Exploration and exploitation are balanced via an epsilon-greedy approach, in which epsilon decays with time to promote more predictable policy execution in subsequent training phases. In 15,000 training instances, the agent engages with an opponent chosen randomly. Key performance metrics are tracked to assess learning progress, including epsilon decay, average rewards, win/draw/loss rates, and training loss. These metrics are visualized to monitor the agent's progress. Human users can compete with the AI by integrating the trained agent with a console-based gaming interface. Simple input forms like "0 0" and "0 1" are accepted by the interface, which shows the board in a comprehensible 3x3 grid. According to the results, the DQN agent gets a high win rate and exhibits significant generalization across different board states. The experiment demonstrates that deep reinforcement learning works well in basic game contexts and provides a basis for extending to more intricate adversarial or multi-agent systems.

Keywords— Deep Q-Network, reinforcement learning, Tic Tac Toe, game AI, Q-learning, epsilon-greedy, experience replay

# Introduction

Artificial intelligence (AI) has advanced significantly in the last ten years, especially in Reinforcement Learning (RL)-based autonomous decision-making. As part of the machine learning discipline of reinforcement learning (RL), an agent learns to make decisions by interacting with its surroundings to maximize cumulative rewards. Trial-and-error learning is the foundation of reinforcement learning (RL), which is particularly well-suited for dynamic situations where agents must gradually acquire the best strategies. This is in contrast to supervised learning, which depends on labelled datasets. One of the most promising methods in reinforcement learning is called Deep Q-learning, which addresses the scalability problems that arise in large or continuous state spaces by combining deep neural networks with conventional Q-learning techniques.

Artificial intelligence has long been tested in games. From the earliest days of algorithms that could play chess to the recent developments in Go and StarCraft, games offer a controlled setting with precise goals and quantifiable results, making them perfect for researching how people learn via play. Although Tic Tac Toe is a basic game compared to contemporary video games, its simplicity, deterministic results, and confined state space make it a popular study topic. It works particularly well as a starting point for testing AI learning techniques.

Due to the small number of possible board configurations, Tic Tac Toe was traditionally solved using traditional game-solving methods like the Minimax algorithm and exhaustive search. Nevertheless, these approaches are not flexible, rely on rules, and cannot extend to more complicated or unpredictable situations. Model-free reinforcement learning algorithms like Q-Learning became increasingly popular as researchers sought more autonomous and scalable methods. An agent can learn the values of state-action pairs using Q-Learning and then use those values to formulate a policy. However, traditional Q-Learning is not practicable in high-dimensional or continuous contexts because it maintains the Q-values in a tabular fashion.

To overcome this constraint, Mnih et al. [1] created Deep Q-networks (DQNs), which approximate the Q-function using a deep neural network. This breakthrough enabled reinforcement learning agents to function in vast and intricate state spaces, such as multi-agent scenarios and visual environments. Because DQNs can learn end-to-end from raw state representations like pictures or numerical vectors and have demonstrated remarkable effectiveness in learning from environments without explicit supervision or human-provided techniques, they offer a substantial development in AI.

The effectiveness of DQNs has been shown in numerous studies in various fields. In the seminal work by Mnih et al. [1], DQNs were applied to Atari 2600 games. In several games, agents beat humans after learning rules directly from pixel inputs. This was the first time deep reinforcement learning was widely used. Later, this method was expanded by DeepMind in their AlphaGo and AlphaZero systems [2], [3], which used DQNs and Monte Carlo Tree Search to learn board games like Go, Chess, and Shogi by self-playing without human assistance. The ability of DQNs to pick up sophisticated tactics and adjust over time is demonstrated by these achievements.

DQNs have been used in real-world applications, including autonomous navigation, robotic control, and games. Deep reinforcement learning was used by Levine et al. [4] to develop visuomotor policies for robotic arms, which allowed them to manipulate objects solely based on visual input. Wei et al. [5] demonstrated the versatility of reinforcement learning in practical systems by introducing IntelliLight. This traffic signal control system uses DQNs to dynamically change signals based on real-time traffic circumstances. The instructional and analytical utility of DQNs has been the subject of numerous studies in simpler settings. For instance, al-Rfou et al. [6] investigated the use of deep reinforcement learning for board games to examine generalization capacities, overfitting, and convergence qualities. Similarly, Zhang et al. [7] used DQNs in multi-agent settings and discovered that cooperative behaviors emerged when agents were trained together. These experiments demonstrate the adaptability of DQNs and their value in fundamental learning contexts such as Tic Tac Toe, where algorithms may be examined, comprehended, and enhanced.

The main goal of this paper is to implement a DQN agent to play Tic Tac Toe. The action-value function is approximated using a feedforward neural network. A flattened 3x3 grid encoded into a nine-dimensional input vector represents the board. An epsilon-greedy strategy, which at first promotes exploration and progressively moves toward exploitation via decaying epsilon with time, is used by the agent to choose actions. To break correlations between successive observations, the model is trained via experience replay, in which previous transitions are randomly selected from a memory buffer. This method increases learning effectiveness and stability. The model's generalization ability is one of the main reasons for using a DQN over conventional Q-Learning. The agent memorizes the precise Q-values for every state-action combination in typical tabular Q-Learning, which causes overfitting in small environments and scalability issues in big ones. Conversely, DQNs allow the agent to estimate action values more accurately for unseen states since they generalize across similar states. As a result, DQNs are more resilient, flexible, and appropriate for uses outside of the Tic Tac Toe toy world.

Likewise, we can see the agent's learning process through various performance measures by putting the DQN into practice in this straightforward setting. Training loss, win/draw/loss rates, average Q-values, reward progression, and epsilon decay are all tracked and visually represented during training. These visualizations shed light on the model's convergence behavior by showing how exploration declines with time and how the agent progressively picks up the best course of action. We can better understand how deep reinforcement learning functions at every level of the training process by incorporating these metrics into our training pipeline for pedagogical and analytical reasons.

We also create an interactive interface allowing human players to compete against the AI to assess the performance of the trained agent. The interface improves usability and user experience by displaying the game board in an organized 3x3 grid style and enabling user inputs in simple formats like "0 0" or "o1". Through this interaction, users with different experience levels with AI systems can be engaged, and the model's performance can be empirically tested.

This work is significant in two ways. Initially, it is a fundamental test for deep reinforcement learning that can be expanded to more intricate settings, such as real-time applications or multi-agent systems. Second, offering a transparent, repeatable, and thoroughly documented use of DQNs in a limited setting benefits the research and teaching communities. These implementations are useful for experimentation, education, and the gradual advancement of AI systems.

With performance tracking, an interactive user interface, and thorough training visualizations, this study presents a full DQN-based agent for Tic Tac Toe. This work lays the groundwork for future investigations into scalable, general-purpose AI systems by expanding on previous research and best practices in reinforcement learning. It also shows the strength and adaptability of deep Q-learning, even in straightforward contexts.

# METHODOLOGY

The process of creating a Deep Q-network (DQN) agent that can play Tic Tac Toe as best it can via reinforcement learning is covered in this section. The method handles the action-value estimation by combining a neural network-based function approximator with conventional Q-learning techniques. Iterative policy improvement, reward-based feedback, and environmental interaction are all part of the training pipeline.

## Problem Formulation

Tic Tac Toe describes a Markov Decision Process (MDP) with a finite number of states, actions, transition probabilities, and rewards. In this deterministic, turn-based, zero-sum environment, two agents alternately place markings (X or O) on a 3x3 grid. The aim is to align three markers diagonally, vertically, or horizontally. During training, a random agent or a rule-based opponent competes with the reinforcement learning agent (the DQN). The agent receives a reward of +1 for a win, -1 for a loss, and 0 for a draw or an illegal move.

## State and Action Representation

A flat vector of nine numbers represents every board state. Agent-occupied cells are shown as 1, opponent-occupied cells as -1, and vacant cells as 0. This encoding ensures the state input remains symmetrical and numerically stable. Similar to the nine cells on the board, nine distinct actions can be taken in the action space. Training masks or severely penalizes invalid actions (i.e., marking a cell that has already been filled).

## Deep Q-Network Architecture

Using a feedforward neural network, the agent approximates the Q-function, which translates states into action values. The network design comprises an input layer with nine neurons (for the board state), two hidden layers with 128 and 64 neurons respectively using ReLU activation, and an output layer with nine neurons (indicating Q-values for each action). The next move is chosen using an epsilon-greedy strategy based on the output.

## Learning Strategy

The DQN uses two essential mechanisms—experience replay and epsilon decay—to guarantee sample efficiency and training stability. Experience replay enables the model to learn from uncorrelated batches of previous experiences by storing transition tuples (state, action, reward, next state, done) in a memory buffer. The epsilon-greedy policy promotes exploration early and exploitation later by starting with strong exploration (ε = 1.0) and linearly decaying to a minimum threshold (e.g., ε = 0.01).

## Training and Optimization

A random sample of experiences is taken from the replay buffer during training, and the mean squared error (MSE) loss between the target and predicted Q-values is used to update the model. The Adam optimizer is employed with a learning rate of 0.001. The target Q-values are calculated using the Bellman equation:

where r is the reward, γ is the discount factor (typically 0.99), and s′ is the next state.

## Training Environment and Evaluation

The agent is trained against a predetermined opponent (random or rule-based) across thousands of episodes. Several indicators, including win/draw/loss rate, reward trends, average Q-values, and training loss, are used to assess performance after training. Graphs are created to illustrate convergence behavior and learning progress.

# Results and Discussion

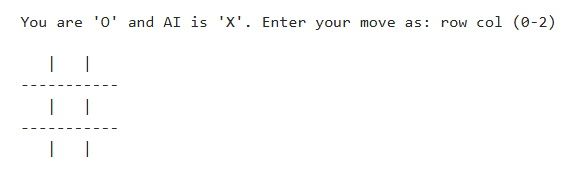


Fig. 1. Normal console for the project

Fig. 1. Show the console of the game. In this game two players will playing this game one is ai and other one is human. Plyer should the give the coordinate position to play this game.

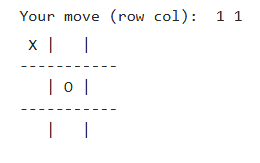


Fig. 2. Playing scenario 1

Fig. 2. shows that user gives his first move which is 1 1.Then the AI agent its move to 0 0.

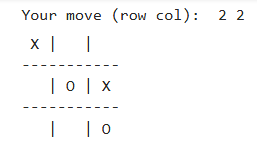


Fig. 3. Playing scenario 2

Fig. 3. shows that user gives his second move which is 2 2.Then the AI agent its move to 1 2.

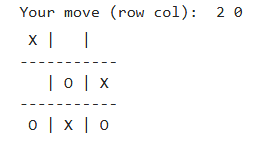


Fig. 4. Playing scenario 3

Fig. 4. shows that user gives his third move which is 2 0.Then the AI agent its move to 2 1.

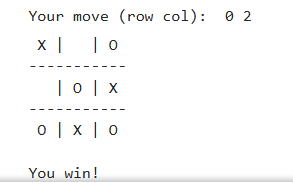


Fig. 5. Playing scenario 4

Fig. 5. shows that user gives his third move which is 1 0. By giving this move user win the match and the game is over.

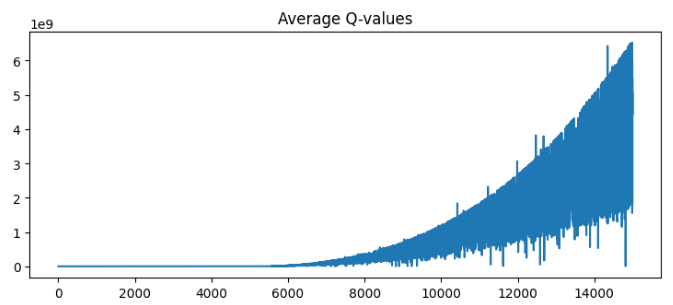


Fig. 6. Average Q-values per Episode during DQN Training

The fig. 6. displays a rapid exponential increase in average Q-values during training, especially after episode 5,000, peaking beyond 10910^9109. This suggests Q-value overestimation, a known issue in DQN due to max-based target updates. Such instability may hinder learning effectiveness and can be addressed using techniques like target networks, Double DQN, or tuning hyperparameters like learning rate.

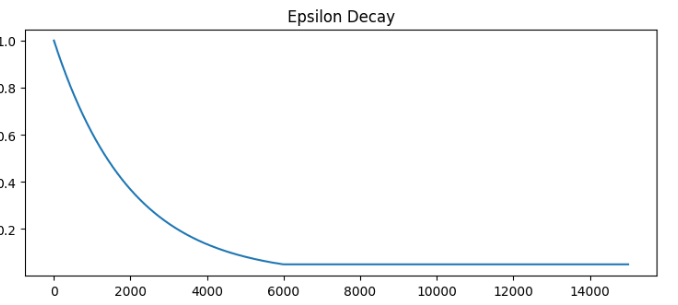


Fig. 7. Epsilon Decay over Episodes

The fig. 7. shows the decreasing trend of the epsilon value throughout the training process. Initially set to 1.0, epsilon gradually decays with each episode, encouraging the agent to shift from exploration to exploitation. Around episode 6000, it reaches the minimum threshold of 0.05 and remains constant. This transition reflects the agent’s increasing confidence in its learned policy over time.

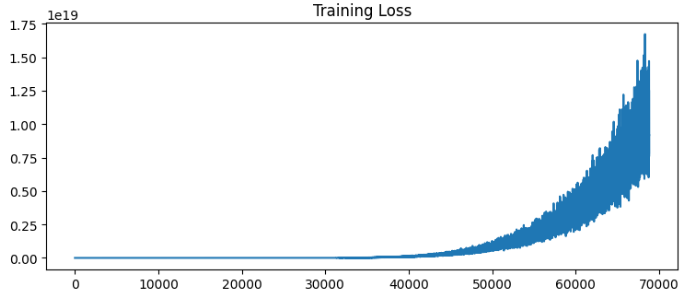


Fig. 8. Training Loss Progression of Deep Q-Network (DQN)

This fig. 8. illustrates the training loss of the DQN agent over time while learning to play Tic Tac Toe. The loss initially remains low but increases significantly in later episodes, indicating instability or divergence during training. This may be due to high Q-values or improper exploration-exploitation balance.

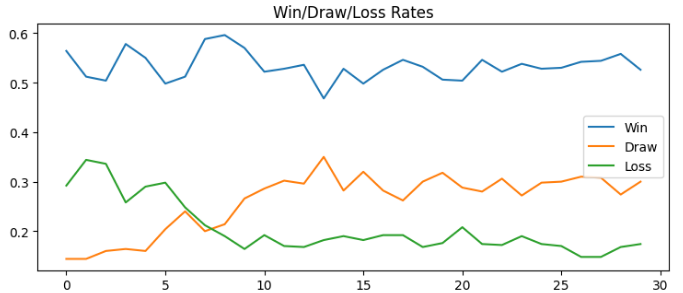


Fig. 9. Win, Draw, and Loss Rates of DQN Agent Over Episodes

This fig. 9. illustrates the win, draw, and loss rates of the DQN agent over 30 evaluation intervals. The win rate remains consistently higher, indicating the agent's improved performance. Meanwhile, the loss rate gradually decreases, showing learning progression. The draw rate fluctuates but generally increases slightly, suggesting more balanced outcomes as the agent becomes more strategic over time.

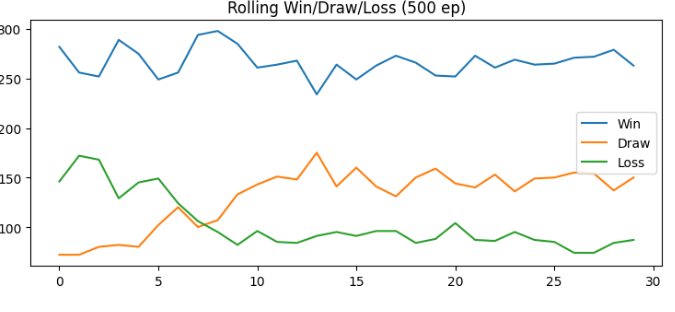


Fig. 10. Rolling Performance over Episodes

The fig. 10. presents the agent’s performance trends throughout training in batches of 500 episodes. It illustrates how often the agent wins, draws, or loses against a random opponent. The blue line, representing wins, remains consistently high, indicating that the agent has successfully learned to play and win most of the time. The orange line shows a gradual increase in draws, suggesting improved defensive play. The green line, indicating losses, declines significantly as training progresses, reflecting the agent’s ability to avoid losing. Overall, the figure demonstrates steady learning and dominance by the trained DQN agent

# Conclusions

We have effectively implemented a Deep Q-Network (DQN) for the game of Tic Tac Toe in this study. The creation of an AI agent that can interact with the gaming environment to learn the best strategies was the main accomplishment of this project. To track the agent's learning progress, we included several training metrics such as epsilon decay, average Q-values, and rewards per episode. Furthermore, incorporating rolling win, draw, and loss counts offered insightful information about the agent's performance over time.

Although the DQN implementation showed encouraging outcomes, several drawbacks were noted. One significant issue was the possibility of overfitting, as Tic Tac Toe's small state space can cause the model to memorize tactics instead of generalizing them. Even if the training period is brief for this simple environment, it might still be a barrier for more complex games. Additionally, the use of a single neural network for Q-value estimation may limit the performance and stability of the learning process. Improvement is needed in a number of areas. A promising strategy that may assist reduce the overestimation bias in the Q-value estimates is the use of Double DQN. In order to improve the agent's capacity to distinguish between action advantages and state values, dueling networks may also be investigated. This could potentially increase learning efficiency. The implementation might also be expanded to accommodate more complicated games or bigger settings, such Connect-4 or variations of chess, where the state space and action space are considerably bigger. Future research will concentrate on expanding the current DQN model to include more intricate game settings and real-world control duties. Putting DQN into practice in settings like Connect-4, chess variations, or even robotics control tasks may offer important new perspectives on the real-world uses of reinforcement learning in decision-making systems. Additionally, investigating more recent hybrid models and reinforcement learning approaches may result in more resilient and effective learning agents that can manage real-time decision-making in dynamic settings.

##### References

1. V. Mnih et al., “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint arXiv:1312.5602*, 2013.
2. D. Silver et al., “Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
3. D. Silver et al., “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.
4. S. Levine et al., “End-to-End Training of Deep Visuomotor Policies,” *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.
5. H. Wei et al., “IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control,” *ACM SIGKDD*, 2018..
6. R. Al-Rfou et al., “Deep Reinforcement Learning for Board Games,” *NIPS Deep Learning Workshop*, 2016.
7. J. Zhang et al., “Learning Multi-Agent Cooperation with Deep Q-Networks,” *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.