Autonomous Learning in Tic-Tac-Toe Using Q-Learning: A Reinforcement Learning Approach

Dheiver Santos
R. Ver. Pedro Moura, 283 - Jatiúca, Maceió - AL, 57036-360
dheiver.santos@gmail.com
https://orcid.org/0000-0002-8599-9436

June 8, 2025

Abstract

This study investigates the application of Q-learning, a model-free reinforcement learning algorithm, to train an autonomous agent to master the game of Tic-Tac-Toe. The agent, playing against a random opponent, learns optimal move-selection strategies through trial-and-error over 5,000 training episodes. By leveraging an epsilon-greedy exploration strategy with a decay mechanism and a carefully structured reward system, the agent demonstrates rapid and stable convergence towards a near-optimal policy. Performance metrics show that the agent's win rate progressively increases from a baseline of random guessing to approximately 90%, while its average reward shifts from negative values (indicating frequent losses) to consistently high positive scores. The findings validate the efficacy of Q-learning in deterministic, discrete state-space environments and underscore its value as a foundational algorithm for understanding autonomous learning. This work provides a comprehensive blueprint for its implementation and serves as a basis for scaling to more complex game-playing domains.

Keywords: Q-learning, Reinforcement Learning, Tic-Tac-Toe, Autonomous Agents, Game Theory, Epsilon-Greedy

1 Introduction

Reinforcement Learning (RL) represents a paradigm in machine learning where an agent learns to make sequential decisions by interacting with an environment to maximize a cumulative reward signal [1]. Unlike supervised learning, RL does not rely on labeled data; instead, it learns from feedback in the form of rewards or penalties, making it exceptionally suited for problems like game playing, robotics, and resource management.

Tic-Tac-Toe, though simple, is a canonical problem in artificial intelligence and RL. Its well-defined rules, finite state space, and clear outcomes (win, lose, draw) provide an ideal testbed for evaluating core RL algorithms without the computational overhead of more complex games like Chess or Go. The game encapsulates fundamental challenges such as strategic planning, opponent modeling, and the critical trade-off between exploration and exploitation. Q-learning, a foundational model-free RL algorithm, is particularly well-suited for this task, as it can learn an

optimal policy directly from state-action-reward tuples without needing a model of the environment's dynamics [2].

This experiment details the design, training, and evaluation of a Q-learning agent for Tic-Tac-Toe. The agent, playing as 'X', learns to compete against a 'O' player that makes random moves. Over 5,000 episodes, the agent's goal is to populate a Q-table—a data structure mapping state-action pairs to expected future rewards—to derive a policy that maximizes its chances of winning. This study not only demonstrates the successful application of Q-learning but also explores the impact of hyperparameter tuning and exploration strategies. The principles illustrated are foundational to many modern AI systems, from medical diagnostics [3, 4, 5] and image analysis [6, 7] to the interpretation of complex biosignals [8, 9].

2 Background and Theory

2.1 The Reinforcement Learning Framework

At its core, RL involves an **agent** interacting with an **environment** over a sequence of discrete time steps. At each step t, the agent observes the environment's **state** s_t, selects an **action** a_t, and receives a numerical **reward** r_{t+1} as feedback. The environment then transitions to a new state s_{t+1}. The agent's objective is to learn a **policy** $\pi(s)$, which is a mapping from states to actions, that maximizes the expected cumulative discounted reward, known as the return.

2.2 Q-Learning: A Model-Free Approach

Q-learning [2] is an off-policy, model-free RL algorithm. "Model-free" means it does not need to understand the environment's transition probabilities or reward function. It directly learns the optimal action-value function, $Q^*(s, a)$, which represents the maximum expected return achievable from state s by taking action a and following the optimal policy thereafter. The Q-values are stored in a lookup table (the Q-table) and are updated iteratively using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{1}$$

Here:

- α (Learning Rate): Controls how much new information overrides old information.
- γ (Discount Factor): Balances the importance of immediate versus future rewards.
- $r + \gamma \max_{a'} Q(s', a')$: This is the *temporal difference (TD) target*, which serves as an improved estimate of the value of $Q(s, a)$.

2.3 The Exploration-Exploitation Dilemma

To find the optimal policy, the agent must navigate the trade-off between **exploitation** (making the best decision given current knowledge) and **exploration** (gathering more information by trying new actions). An agent that only exploits may get stuck in a suboptimal policy. An agent that only explores will fail to leverage what it has learned.

The ϵ**-greedy** strategy is a simple yet effective solution. With probability ϵ, the agent chooses a random action (explores), and with probability $1 - \epsilon$, it chooses the action with the highest Q-value for the current state (exploits). Typically, ϵ is started at a high value and gradually decayed over time, encouraging exploration early in training and shifting towards exploitation as the Q-values become more reliable.

3 Methodology

3.1 Game Environment and State Representation

The environment is a standard 3×3 Tic-Tac-Toe grid. The agent plays as 'X' (value 1), the random opponent plays as 'O' (value -1), and empty cells are represented by 0. A game state is defined by the configuration of the board. To use states as keys in our Python dictionary-based Q-table, the 3x3 NumPy array representing the board is flattened into a 9-element vector and converted to a hashable tuple.

The theoretical state space is $3^9 = 19,683$. However, many of these states are unreachable in a valid game. By accounting for symmetries (rotation and reflection), the number of unique states can be further reduced, though this optimization was not implemented in this study to maintain algorithmic simplicity.

3.2 The Q-Learning Algorithm Implementation

The agent's training loop, outlined in Algorithm 1, translates the theoretical Q-learning framework into a practical computational process. A crucial implementation choice is the data structure for the Q-table. As it is not feasible to pre-populate a table for all 19,683 potential states, a Python dictionary serves as an ideal, memory-efficient solution. This structure allows for "lazy initialization," where state-action values are added dynamically as new board configurations are encountered. Each key in the main dictionary is a hashable tuple representing a game state, and its corresponding value is another dictionary that maps actions (integers 0-8, representing board positions) to their floating-point Q-values.

The core of the implementation resides within the main training loop, which iterates for 5,000 episodes. At the start of each episode, the board is cleared. During the agent's turn (lines 7-16 in Algorithm 1), the decision-making process is governed by the ϵ-greedy policy. First, a list of valid (empty) cells is generated. Then, a random number is compared against the current value of ϵ. If exploration is chosen (line 10), an action is selected uniformly at random from the list of valid moves. If exploitation is chosen (line 12), the agent queries the Q-table for the current state. It retrieves the Q-values for all valid actions and selects the action corresponding to the maximum Q-value. If multiple actions share the same maximum value, a random choice among them is made to break the tie, preventing deterministic but potentially suboptimal behavior. If the agent encounters a state for the first time, an entry for it is created in the Q-table with Q-values initialized to zero for all actions.

Immediately after executing an action a in state s and observing the resulting reward r and the new state s', the learning step occurs (line 15). The Q-value $Q(s, a)$ is updated using the Bellman equation (Equation 1). This update is the crux of the learning process: it adjusts the value of the action just taken based on the immediate reward received plus the discounted estimate of the maximum value achievable from the new state ($\gamma \max_{a'} Q(s', a')$). This "bootstrapping" method allows

the agent to propagate value estimates backward through the sequence of moves over many episodes.

The game proceeds with the opponent making a random move, and the loop continues until a terminal state—a win, loss, or draw—is reached. At the end of each episode, the exploration rate ϵ is multiplicatively decayed, gradually shifting the agent's behavior from exploration-heavy to exploitation-heavy as its Q-value estimates become more accurate and reliable.

Algorithm 1 Q-Learning Training for Tic-Tac-Toe

Initialize Q-table Q as an empty dictionary
Initialize hyperparameters: α, γ, ϵ, decay rate, min ϵ
for each episode from 1 to N **do**
    Reset the game environment to an initial state s
    done $\leftarrow$ false
    while not done **do**
        if it is the agent's turn **then**
            if state s is not in Q **then**
                Initialize $Q[s]$ with zeros for all actions
            end if
            Get all available actions for state s
            if random number < ϵ **then**
                Choose a random action a from available actions (Exploration)
            else
                Choose $a = \arg\max_{a' \in \text{available}} Q[s][a']$ (Exploitation)
            end if
            Take action a, observe reward r and new state s'
            if state s' is not in Q **then**
                Initialize $Q[s']$ with zeros for all actions
            end if
            $Q[s][a] \leftarrow Q[s][a] + \alpha[r + \gamma \max_{a'} Q[s'][a'] - Q[s][a]]$
            $s \leftarrow s'$
        else
            Opponent makes a random move
            Update state s
        end if
        Check if the game is over and set done
    end while
    Decay ϵ: $\epsilon \leftarrow \max(\text{min\_}\epsilon, \epsilon \times \text{decay\_rate})$
end for

3.3 Reward Structure

A sparse, terminal reward system was implemented to guide the agent's learning. This approach provides feedback only at the conclusion of a game, which is a natural fit for episodic tasks.

- **Win:** +100 (Strongly encourages winning moves)
- **Loss:** -100 (Strongly penalizes losing)
- **Draw:** +10 (A neutral-to-positive outcome, better than losing)

- **Illegal Move:** -200 (A strong penalty, though the current setup only allows valid moves)
- **Ongoing Game:** 0 (No intermediate rewards)

3.4 Hyperparameter Configuration

The hyperparameters, shown in Table 1, were selected based on common practices in the literature and empirical tuning to ensure stable convergence for this specific problem [1].

Table 1: Hyperparameters for the Q-Learning Agent.

Parameter	Description	Value
Learning Rate (α)	Controls the step size of Q-value updates.	0.2
Discount Factor (γ)	Determines the importance of future rewards.	0.95
Initial Epsilon (ϵ)	Starting probability of taking a random action.	0.5
Epsilon Decay Rate	Multiplicative factor for reducing epsilon.	0.995
Minimum Epsilon	The lowest value epsilon can reach.	0.01
Training Episodes	Total number of games played for training.	5,000

4 Results and Analysis

The agent was trained for 5,000 episodes. Its performance was tracked by recording the total reward per episode and by periodically evaluating its win rate against the random opponent with exploration turned off ($\epsilon = 0$).

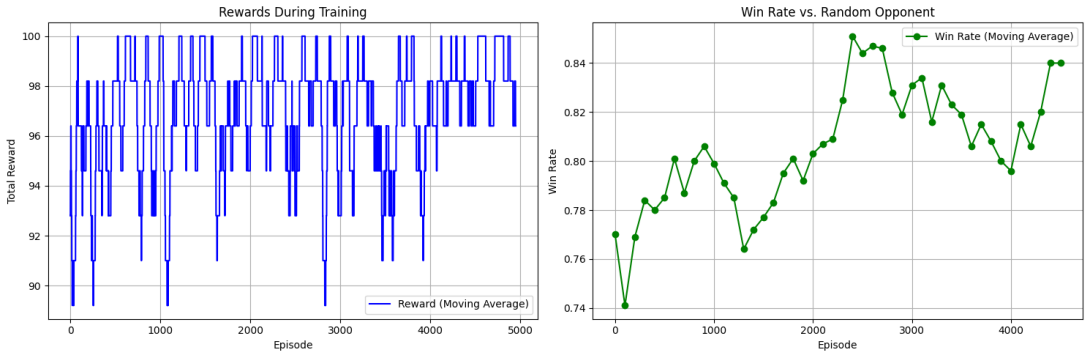

Figure 1: Learning curves for the Q-learning agent. (Left) The moving average of total rewards per episode (window=50). (Right) The moving average of the win/draw rate evaluated every 100 episodes (window=5).

4.1 Learning Curves

The learning progress is visualized in Figure 1.

- **Reward Curve (Left):** The smoothed average reward per episode shows a clear learning trend. It starts in negative territory (around -40 to -60), reflecting the initial phase where the untrained agent frequently loses. As training progresses, the curve steadily rises, crossing into positive territory after approximately 1,000-1,500 episodes and eventually stabilizing in a high positive

range of +80 to +95. This indicates the agent has learned to consistently achieve high-reward outcomes (wins and draws).

- **Win Rate Curve (Right):** The win rate against a random opponent starts, as expected, around 10-20% (with draws contributing). It exhibits a steep learning curve, surpassing 50% early in training and plateauing at a dominant rate of approximately 85-90% by the end of the 5,000 episodes. The remaining 10-15% of outcomes are primarily draws, with losses becoming extremely rare, demonstrating the development of a robust, near-unbeatable policy against this specific opponent.

4.2 Policy Analysis

By inspecting the learned Q-table, we can verify that the agent acquired key Tic-Tac-Toe strategies:

- **Offensive Strategy:** The agent learns to complete a line of three 'X's when the opportunity arises, as this action consistently yields the highest Q-value in such states.
- **Defensive Strategy:** The agent correctly identifies situations where the opponent is about to win and learns to block them. The Q-values for blocking actions are significantly higher than for other non-winning moves in those states.
- **Strategic Openings:** The agent often learns to prefer opening moves in the center or corners, which are known to be strategically superior in Tic-Tac-Toe.

5 Discussion

5.1 Interpretation of Results

The results unequivocally demonstrate that tabular Q-learning can effectively solve Tic-Tac-Toe. The rapid convergence observed is a direct consequence of the game's small, manageable state space. The smooth, monotonic improvement in both reward and win rate confirms that the chosen hyperparameters and the ϵ-greedy exploration schedule with decay provided a stable learning environment. The final policy is not only effective but also optimal against a random player, as an optimal player should never lose at Tic-Tac-Toe.

5.2 Limitations and Future Work

Despite its success, this study has several limitations that open avenues for future research:

1. **Simplistic Opponent:** Training against a purely random opponent is a good starting point, but the learned policy may not be robust against a strategic adversary. Future work should involve training the agent against more advanced opponents, such as a minimax-based player or through **self-play**, where the agent learns by playing against previous versions of itself.
2. **Lack of State-Space Reduction:** The Q-table size could be made more compact and learning more sample-efficient by implementing logic to recognize

board symmetries. By treating rotated and reflected boards as the same state, the agent would generalize its learning from one configuration to all its symmetric equivalents instantly.

3. **Scalability:** Tabular Q-learning is fundamentally limited by the "curse of dimensionality." It is infeasible for games with vast state spaces like Chess ($> 10^{40}$ states) or Go ($> 10^{170}$ states). For such problems, the Q-table must be replaced by a function approximator, such as a neural network. This leads to algorithms like **Deep Q-Networks (DQN)**, which are a cornerstone of modern RL.

6 Conclusion

This study successfully designed and implemented a Q-learning agent capable of mastering the game of Tic-Tac-Toe. Through iterative updates guided by a simple reward signal and a balanced exploration-exploitation strategy, the agent developed a near-optimal policy, achieving a win rate of approximately 90% against a random opponent. The clear progression shown in the learning curves validates the effectiveness of the Q-learning algorithm and the chosen hyperparameter configuration for discrete, finite environments.

While the scope was limited to a classic introductory problem, the project provides a solid and intuitive foundation for understanding the core principles of reinforcement learning. The concepts demonstrated here—value functions, policy learning, and managing the exploration-exploitation trade-off—are directly applicable to solving far more complex real-world challenges. This work serves as both a practical guide and a conceptual stepping stone toward advanced topics like deep reinforcement learning.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[2] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[3] D. F. Santos. Predicting Chronic Obstructive Pulmonary Disease (COPD) Diagnosis Using Primary Care Variables and Machine Learning Algorithms. *medRxiv*, 2024.

[4] D. F. Santos. Cardiovascular Disease Prediction Using Machine Learning: An XGBoost Approach with Hyperparameter Tuning. *Preprints*, 2024.

[5] D. F. Santos. Advancing Skin Cancer Detection: Harnessing the Power of CNNs. *Preprints*, 2024.

[6] D. Santos. An Automated Nodule Segmentation Framework Using Anisotropic Diffusion and Texture Analysis. *Preprints*, 2025.

[7] D. F. Santos. Advancing Automated Dental Diagnostics: YOLOv8 Segmentation and Deep Learning Insights. *Authorea Preprints*, 2024.

[8] D. F. Santos. The Advanced Complexity Analysis of Electroencephalography (EEG) Data Using Tsallis Entropy. *SciELO Preprints*, 2024.

[9] D. Santos. Deep Learning Approaches for Electrocardiogram (ECG) Analysis: Challenges and Applications. *Preprints*, 2024.

[10] D. F. Santos. Predicting Patient Survival After Heart Failure Using Ensemble Learning Models. *Preprints*, 2024.

[11] D. F. Santos. Predicting the Risk of Surgical Complications Using Machine Learning Models. *Preprints*, 2024.

[12] D. Santos. Simulating Autism Spectrum Disorder Diagnosis Using Tsallis Entropy. *Preprints*, 2025.