Training The Classic Snake Game: Benchmarking Q-Learning vs DQN

Diana Nicole Danga
College of Computing and Information Technologies
National University
Manila, Philippines
dangadd@students.national-u.edu.ph

John Efren Gannaban
College of Computing and Information Technologies
National University
Manila, Philippines
gannabanjv@students.national-u.edu.ph

Jascent Pearl Navarro
College of Computing and Information Technologies
National University
Manila, Philippines
navarrojg@students.national-u.edu.ph

***Abstract*—This study compares Q-Learning and Deep Q-Networks (DQN) in a discrete grid environment using a comprehensive multi-metric evaluation framework. Agents were assessed on sample efficiency, convergence stability, exploration stability, policy behavior, computational efficiency, and post-training greedy policy performance across multiple seeds. Q-Learning achieves rapid initial learning with low computational cost but exhibits instability, catastrophic forgetting, high reward variance, and sensitivity to coarse state representations. DQN demonstrates smoother convergence, reduced reward variance, and improved inter-seed consistency due to neural function approximation, target networks, and experience replay, though early-stage learning is slower and computationally costlier. Results clarify how state representation, exploration scheduling, and learning dynamics affect reinforcement learning outcomes, highlighting trade-offs between tabular and neural approaches. Findings emphasize the need for richer state encodings, adaptive learning rates, and extended training. Future work includes deeper architectures, improved exploration schedules, and advanced techniques such as prioritized experience replay and double Q-learning to enhance policy stability and near-optimal performance.**

***Index Terms*—Reinforcement Learning, Q-Learning, DQN, Benchmarking, Snake Game, Comparative Analysis**

I. Introduction

Reinforcement Learning (RL) has emerged as a fundamental paradigm for developing autonomous agents capable of learning optimal decision-making strategies through environmental interaction [1]. Among various benchmark environments used to evaluate RL algorithms, game-based scenarios offer controlled yet complex settings that exhibit key challenges present in real-world applications [2]. The Snake game, despite its apparent simplicity, presents a compelling testbed for RL research due to its dynamic state space, sparse reward structure, and requirement for long-term strategic planning.

The application of RL to the Snake game domain presents significant technical challenges. As the snake grows longer throughout gameplay, the state space expands exponentially, creating difficulties for learning algorithms in terms of both convergence speed and memory efficiency [3]. Additionally, the sparse and delayed reward structure where positive feedback occurs only upon food consumption complicates the credit assignment problem inherent to sequential decision-making tasks [4]. These characteristics make the Snake game an appropriate benchmark for evaluating the practical effectiveness of different RL approaches.

Existing literature has explored various RL methodologies across different game environments, yet comprehensive comparative studies within standardized Snake game implementations remain limited [5]. While theoretical advancements in RL continue to emerge, empirical validation through systematic comparison is essential for understanding the practical trade-offs between algorithmic complexity and performance gains.

This paper presents a comparative study of two foundational RL approaches, Q-Learning and Deep Q-Networks (DQN) applied to the Snake game environment. While prior studies primarily report terminal metrics such as final score and convergence rate, this work extends the analysis to include learning stability, sample efficiency, exploration behavior, and computational efficiency. Specifically, we track performance variance, episode length consistency, and convergence smoothness across multiple runs, while also examining reward growth relative to training episodes, exploration decay versus reward variance, runtime efficiency, and policy behavior evolution.

The main contributions of this work include the implementation and evaluation of Q-Learning and DQN agents within a standardized Snake game framework using consistent evaluation metrics. We provide a comparative analysis of learning stability, sample efficiency, exploration consistency, convergence behavior, and runtime performance alongside conventional metrics. Finally, we offer diagnostic insights into learning dynamics through variance analysis, convergence tracking, interpretive learning curves, and qualitative policy behavior assessment. By emphasizing how learning unfolds rather than only final outcomes, this work provides a transparent and practical understanding of reinforcement learning performance in the Snake environment.

II. Literature Review

A. Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with how autonomous agents learn to make sequential decisions through interaction with an environment to

maximize cumulative rewards [6]. The fundamental framework of RL is typically formalized as a Markov Decision Process (MDP), defined by a tuple $(S, A, R, P\gamma)$, where S represents the set of states, A the set of actions, R the reward function, the transition probability between states, and γ the discount factor determining the importance of future rewards [7]. The agent's objective is to learn an optimal policy $\pi^*(a \mid s)$ that maximizes the expected discounted return $E\left[\sum_t \gamma^t r_t\right]$ through iterative interaction with the environment.

In this paradigm, two key approaches have emerged: value-based methods and policy-based methods. Value-based methods, such as Q-Learning, aim to estimate the optimal action-value function $Q(s, a)$, which represents the expected cumulative reward of taking action a in state s and following the optimal policy thereafter [8]. Policy-based methods, on the other hand, directly optimize the policy without estimating intermediate value functions. Among value-based methods, Deep Reinforcement Learning (DRL) has gained prominence due to its ability to approximate complex value functions using deep neural networks. This paradigm shift, initiated by the Deep Q-Network (DQN) of Mnih et al. [9], enabled RL agents to perform at human-level competence across high-dimensional environments such as Atari games.

In the context of the Snake game, RL provides an intuitive formulation where the agent (snake) interacts with a grid-based environment, choosing directional actions (up, down, left, right) based on state observations to maximize its cumulative reward, typically defined as the snake's length or score. This environment's sequential decision-making nature, delayed rewards, and dynamic constraints make it a classic benchmark for testing the learning efficiency and stability of RL algorithms such as Q-Learning and DQN.

B. Q - Learning

Reinforcement Learning (RL) has emerged as a powerful paradigm for developing autonomous agents capable of learning optimal strategies through environmental interaction. Among RL algorithms, Q-Learning is a foundational, model-free, value-based method that enables agents to learn the expected reward for actions in a given state. The Q-Learning algorithm updates the action-value function $Q(s, a)$ using the Bellman equation: $Q(s, a) \leftarrow Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$ where α is the learning rate, r the immediate reward, γ the discount factor, s' the next state, and a' the next action [10]. In the context of the Snake game, Q-Learning allows the agent to learn optimal movement strategies by associating state-action pairs with expected future rewards.

Existing approaches using Q-Learning in the Snake game demonstrate both strengths and limitations. Q-Learning's tabular approach is computationally light and conceptually simple, but it struggles with scalability as the state space grows exponentially with the size of the grid and the snake's length. Moreover, balancing exploration and exploitation can be challenging, often leading to suboptimal policies. Overfitting has also been observed in some implementations, resulting in decreased performance after extensive training [11].

Deep Q-Networks (DQN) have been proposed to address these limitations by combining Q-Learning with deep neural networks to approximate Q-values. DQN enables handling of large and complex state spaces, improving the agent's ability to learn effective strategies in environments like Snake. However, it introduces new challenges, including overfitting, training instability, and increased computational demands. Techniques such as experience replay and target networks are often employed to stabilize training [12].

Compared to traditional Q-Learning, DQN offers significant improvements in state representation and policy performance. While Q-Learning uses tabular methods that limit scalability, DQN leverages neural networks for function approximation, allowing it to manage larger state spaces and improve exploration efficiency. Nonetheless, both approaches remain model-free, value-based, and off-policy, as they do not require knowledge of the environment dynamics and focus on estimating the optimal action-value function.

While Q-Learning provides a foundational approach to reinforcement learning in the Snake game, its limitations in scalability and overfitting are notable. DQN improves upon these issues through deep learning, although challenges such as training instability and computational intensity persist. Hybrid or advanced RL techniques may further enhance performance, making them promising directions for future research.

C. Deep Q-Network (DQN)

The Deep Q-Network (DQN) algorithm represents a significant milestone in the advancement of reinforcement learning by integrating the principles of Q-learning with the representational capabilities of deep neural networks. Introduced by Mnih et al. [13], DQN demonstrated that a single neural network architecture could learn to play various Atari 2600 games directly from raw pixel inputs, achieving near-human or superior performance levels [14]. The algorithm addresses one of the key limitations of classical Q-learning, namely the inability to scale to large or continuous state spaces, by approximating the action-value function $Q(s, a)$ through a deep neural network parameterized by weights θ.

The objective of DQN is to minimize the temporal-difference error between the predicted Q-values and the target Q-values, which are computed using a separate target network with frozen parameters θ^- to enhance training stability. Moreover, DQN employs an experience replay buffer that stores past transitions (s, a, r, s'), allowing the agent to sample mini-batches of experiences uniformly and reduce correlations between consecutive updates. These innovations collectively improved the convergence stability and sample efficiency of value-based reinforcement learning methods.

Despite its pioneering success, the original DQN exhibited key limitations such as overestimation bias, sample inefficiency, and sensitivity to hyperparameter settings, which motivated subsequent refinements. Van Hasselt et al. [15] introduced Double DQN, decoupling action selection and evaluation to reduce overoptimistic Q-value estimates and improve policy stability across Atari benchmarks. To further address inefficiencies in environments with redundant or similar actions, Wang et al. [16] proposed the Dueling DQN architecture, which separates state value and action advantage representations to enhance sample efficiency and generalization. Building upon these advances, Hessel et al. [17] unified multiple extensions including Double DQN, Dueling architecture, prioritized experience replay, multi-step learning, noisy nets, and distributional reinforcement learning into the Rainbow DQN, achieving state-of-the-art, stable, and sample-efficient performance across Atari 2600 benchmarks.

In the context of game environments, DQN has been widely applied to benchmark tasks that require spatial awareness, sequential decision-making, and delayed rewards characteristics that make it highly suitable for the Snake Game. The Snake Game provides a controlled yet challenging environment in which an agent must navigate a grid, collect food items, and avoid collisions, leading to a dynamic balance between exploration and exploitation. Traditional tabular Q-learning approaches struggle with this task due to the exponential growth of possible states as the snake lengthens, making function approximation through deep networks a more scalable alternative.

More recent studies have applied DQN and variants directly to Snake; Sebastianelli et al. [18] evaluated hyperparameter tuning of DQN models on Snake; Tushar & Siddique [19] proposed a memory-efficient CNN-based variant of the Q-network to reduce resource demands; and Pan et al. [20] identified overfitting and environment-information dependency when using DQN in Snake. These works confirm that DQN remains one of the most widely used algorithms for training RL agents in Snake and underscore practical aspects such as hyperparameter sensitivity, reward design, and function approximation stability, while also revealing opportunities to explore performance beyond conventional accuracy and convergence measures.

D. Synthesis

Although both Q-Learning and Deep Q-Networks (DQN) have been extensively explored in the Snake game environment, existing literature consistently exhibits methodological gaps that constrain deeper understanding of agent learning behavior and efficiency. Foundational works by Mnih et al. and subsequent refinements such as Double DQN, Dueling DQN, and Rainbow established the stability and scalability of deep reinforcement learning across Atari benchmarks. Yet, when adapted to Snake environments, studies have largely centered on terminal metrics such as final score, maximum length, and convergence rate while overlooking secondary but critical factors such as sample efficiency, computational cost, and exploration stability.

Traditional Q-Learning implementations in Snake, while computationally lightweight and conceptually accessible, face fundamental scalability challenges that remain inadequately addressed in current literature. The exponential growth of the state space with grid size and snake length renders tabular Q-Learning impractical for larger environments, yet few studies systematically quantify the threshold at which performance degrades or memory requirements become prohibitive. Moreover, the exploration-exploitation trade-off in Q-Learning is often managed through simple ϵ-greedy strategies without rigorous analysis of how exploration schedules affect convergence speed, policy stability, or susceptibility to local optima. The overfitting phenomenon observed in extended Q-Learning training sessions suggests unstable learning dynamics that warrant deeper investigation through variance analysis and cross-validation approaches, yet such diagnostic methods are rarely employed.

These omissions have made most Snake RL studies whether using Q-Learning or DQN demonstrative rather than diagnostic: they show that algorithms work, but not how efficiently, robustly, or sustainably they learn. In particular, sample efficiency is often left unquantified. For example, Tushar and Siddique trained their memory-optimized DQN for 140,000 episodes without analyzing the minimum episode count needed for competent play, while Q-Learning studies similarly lack episode-efficiency benchmarks that would clarify when tabular methods remain viable versus when function approximation becomes necessary. Similarly, computational and energy costs remain unmeasured despite broader discussions on the sustainability of deep learning and the often-cited advantage of Q-Learning's lower computational overhead, an advantage that has not been empirically validated in comparative studies. Exploration behavior is also rarely analyzed beyond ϵ-greedy decay schedules in both paradigms, leaving open questions about how different exploration dynamics influence stability or early plateauing. Finally, generalization and hyperparameter robustness have received minimal attention, with few studies examining transfer across environment variations, comparing sensitivity between Q-Learning's α and γ versus DQN's network architecture and replay buffer parameters, or performing statistical validation over multiple random seeds.

The study focuses on achievable yet meaningful explorations that provide immediate value within these open areas. Specifically, we extend the comparative analysis between Q-Learning and DQN by examining learning stability, sample efficiency, and exploration consistency through lightweight yet informative metrics. Rather than introducing new architectures or algorithmic variants, we analyze how learning unfolds in both tabular and function approximation contexts through tracking performance variance, episode length consistency, convergence smoothness, and computational efficiency across runs. These evaluations reveal nuanced behavioral differences

such as Q-Learning's early convergence versus DQN's potential for higher asymptotic performance, or the relative stability-efficiency trade-offs between methods often omitted in prior works, offering practical insights without heavy computational demand. By incorporating variance analysis, simple sensitivity checks, comparative resource profiling, and interpretive learning curves, we contribute to a more transparent and diagnostic understanding of reinforcement learning performance in Snake.

III. Methodology

A. Environment and Problem Formulation

a) Environment or Simulation Used

The experiment was conducted using the Snake-Gym environment, a reinforcement learning simulation based on OpenAI Gym and implemented with Pygame, originally developed by Vivek3141 [21]. This environment was customized to modify its reward structure and wall behavior, allowing the agent to receive more meaningful feedback and learn more efficiently.

In this setting, the agent controls a snake that navigates a two-dimensional grid, attempting to eat apples that appear randomly on the screen. Each time the snake consumes an apple, its body grows in length. The episode terminates if the snake collides with the wall or with its own body. The custom modifications removed the original "wrap-around" behavior (where the snake could exit one side of the screen and reappear on the other) and instead enforced solid boundary collisions, making the environment more challenging and realistic for reinforcement learning.

b) State Space and Action Space

State Space: The environment's state space is represented as a 150×150 grid, corresponding to the pixel-based game screen. Each grid cell encodes one of several possible states:

- $0 \rightarrow$ Empty space
- $1 \rightarrow$ Snake's body
- $2 \rightarrow$ Snake's head
- $3 \rightarrow$ Apple

These values form the observation returned to the agent, allowing it to perceive the full layout of the environment. Internally, the game also tracks positional information such as:

- The (x, y) coordinates of the snake's head
- The coordinates of the apple
- The positions of the snake's body segments
- The current movement direction (UP, DOWN, LEFT, or RIGHT)

This structured state representation provides the agent with sufficient spatial information to navigate effectively, identify targets, and avoid collisions.

Action Space: The environment defines a discrete action space with four possible actions:

- $0 \rightarrow$ Move Up
- $1 \rightarrow$ Move Down
- $2 \rightarrow$ Move Left
- $3 \rightarrow$ Move Right

Each action corresponds to a directional change in the snake's head position, which directly determines the next movement of the snake within the grid.

c) Reward Function

The reward function was modified from the original Snake-Gym design to encourage efficient learning and discourage unsafe or idle behavior. The following structure was used:

Event	Reward	Description
Eats an apple	+10	Strong positive reinforcement for achieving the goal
Collides with wall or body	-10	Strong penalty for terminal mistakes
Moves closer to apple	+0.1	Small positive reward for progress toward the target
Moves away from apple	-0.05	Small penalty for inefficient movement
Takes a time step	-0.01	Slight penalty to promote faster food collection

This reward system balances exploration and exploitation, helping the agent learn to navigate efficiently toward the apple while minimizing unnecessary movements and avoiding fatal collisions.

d) Terminal Condition

An episode ends when the snake:

- Collides with the wall (due to the removal of the wrap-around feature), or
- Runs into its own body.

No maximum step limit is currently enforced, meaning the episode continues indefinitely until a collision occurs. Upon termination, the environment resets with a newly randomized snake and apple position.

e) Task Type

The Snake task is episodic in nature. Each episode begins with a freshly initialized environment and ends once a terminal condition is met. The agent's objective is to maximize the cumulative reward per episode by developing an optimal policy that balances survival, movement efficiency, and food collection. The episodic structure allows the agent to learn from repeated independent runs, gradually improving its performance across episodes.

B. Algorithm Description

a) Reinforcement Learning Algorithms Used

Two classical reinforcement learning (RL) algorithms were implemented for the Snake-Gym environment: Q-Learning and Deep Q-Network (DQN).

- Q-Learning is a model-free, value-based RL algorithm that learns the optimal state-action value function Q(s,a) iteratively through temporal difference updates. It is widely applied to discrete, low-dimensional state spaces such as grid-world problems.
- DQN is an extension of Q-Learning that approximates the Q-function using a neural network, enabling it to handle high-dimensional state spaces and image-like inputs. DQN incorporates experience replay and a separate target network to stabilize learning.

These two algorithms were chosen to compare classical tabular methods with deep RL approaches and to evaluate their sample efficiency, exploration stability, computational efficiency, convergence stability, and policy learning behavior in a controlled Snake environment.

b) Key Equations or Updates

Q-Learning Key Equations and Implementation

1. Q-Learning Update Rule

The main update for each state-action pair $Q(s, a)$ is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Where:

- s_t = current state
- a_t = action taken in state s_t
- r_{t+1} = reward received after taking action a_t
- s_{t+1} = next state
- α = learning rate (alpha = 0.1 in code)
- γ = discount factor (gamma = 0.95)
- $\max_{a'} Q(s_{t+1}, a')$ = estimate of optimal future value from next state

Code Implementation:

```
# Q-learning update
Q[state][action] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[state][action])
```

2. Epsilon-Greedy Action Selection

Actions are chosen to balance exploration vs. exploitation:

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

Code Implementation:

```
# ε-greedy action selection
if random.random() < epsilon:
    action = random.choice(range(n_actions))
else:
    action = np.argmax(Q[state])
```

Epsilon decays over time to shift gradually from exploration to exploitation:

$$\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon \cdot \epsilon_{\text{decay}})$$

Code Implementation:

```
epsilon = max(epsilon_min, epsilon * epsilon_decay)
```

3. State Initialization (if unseen)

Whenever a new state is encountered, its Q-values are initialized to zeros for all possible actions.

This ensures the agent can handle unseen situations without errors.

$$Q(s_{new}, a) = 0, \quad \forall a$$

Code Implementation:

```
Q = defaultdict(lambda: np.zeros(n_actions))
```

DQN Key Equations and Implementation

1. DQN Q-Value Update (Loss Function)

The core of DQN is minimizing the difference between predicted Q-values and target Q-values:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_{t+1}, s_{t+1}) \sim \text{ReplayBuffer}} \left[\left(r_{t+1} + \gamma \max_{a'} Q_{\text{target}}(s_{t+1}, a'; \theta^-) - Q_{\text{policy}}(s_t, a_t; \theta) \right)^2 \right]$$

Where:

- s_t = current state
- a_t = action taken in state s_t
- r_{t+1} = reward received after taking action a_t
- s_{t+1} = next state
- α = learning rate (e.g., 0.1 in code)
- γ = discount factor (e.g., 0.95)
- $\max_{a'} Q(s_{t+1}, a')$ = estimate of optimal future value from next state

Code Implementation:

```
q_values = policy_net(s_t).gather(1, a_t.unsqueeze(1)).squeeze(1)
with torch.no_grad():
    next_q = target_net(s2_t).max(1)[0]
    target_q = r_t + gamma * next_q * (1 - d_t)
loss = nn.MSELoss()(q_values, target_q)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

2. Epsilon-Greedy Action Selection

DQN uses the same exploration-exploitation strategy as Q-Learning:

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q_{policy}(s_t, a; \theta) & \text{with probability } 1 - \epsilon \end{cases}$$

Code Implementation:

```
# ε-greedy action selection
if random.random() < epsilon:
    action = random.randrange(n_actions)
else:
    with torch.no_grad():
        st = torch.tensor(state_arr, dtype=torch.float32, device=device).unsqueeze(0)
        qvals = policy_net(st)
        action = int(torch.argmax(qvals).item())
```

3. Experience Replay

DQN stores past transitions to break correlation between consecutive samples and improve stability:

$$(s_t, a_t, r_{t+1}, s_{t+1}, done) \in \text{ReplayBuffer}$$

Code Implementation:

```
# store transition
memory.push(state_arr, action, reward, next_state_arr, done)

state_arr = next_state_arr
total_reward += reward
steps += 1

# DQN update (sample mini-batch)
if len(memory) >= batch_size:
    s_batch, a_batch, r_batch, s2_batch, d_batch = memory.sample(batch_size)
```

4. Target Network Update

To stabilize learning, the target network is periodically updated from the policy network:

$$\theta^{-} \leftarrow \theta \quad \text{every } N_{\text{target}} \text{ episodes}$$

Where:

- θ = weights of the policy network
- $\theta-$ = weights of the target network
- N_{target} = number of episodes between updates (target_update in code)

Code Implementation:

```
# Update target network periodically
if (episode + 1) % target_update_freq == 0:
    target_net.load_state_dict(policy_net.state_dict())
```

c) Hyperparameters Used

In the Q-Learning implementation for the Snake environment, several hyperparameters determine how the agent learns from interaction with the environment. The number of training episodes is set to 3,000, allowing the agent sufficient experience to converge toward an optimal policy. The learning rate (α) is 0.1, controlling how strongly new experiences update existing Q-values. The discount factor (γ) is 0.9, determining how much the agent values future rewards compared to immediate ones. The exploration rate (ϵ) starts at 1.0, ensuring high exploration early in training, and decays by 0.995 per episode until it reaches a minimum of 0.05, shifting gradually toward exploitation of learned behavior. These parameters collectively define the agent's learning dynamics, balancing exploration, exploitation, and convergence stability across 3,000 episodes.

In the DQN implementation for the Snake environment, the hyperparameters mirror Q-Learning's structure but extend it to a neural network setup. The number of training episodes is 3,000, maintaining consistency with the tabular approach. The learning rate (*lr*) is set to 0.001, controlling how much the neural network's weights are updated at each step using the Adam optimizer. The discount factor (γ) is 0.9, matching Q-Learning for fair comparison. The exploration rate (ϵ) starts at 1.0, decays by 0.995 per episode, and is bounded by a minimum of 0.05, maintaining the same exploration strategy as Q-Learning.

DQN also introduces several additional hyperparameters:

- Batch size: 64 — number of experiences sampled from the replay buffer per update.
- Replay buffer capacity: 10,000 — maximum number of stored transitions for experience replay.
- Target network update frequency: every 100 episodes — to stabilize learning by providing fixed Q-targets for a period of time.
- Checkpoint frequency: every 1,000 episodes — saves model progress for recovery and analysis.
- Hidden layer size: 64 neurons — chosen for a small MLP to keep complexity comparable to the Q-table approach.

Together, these hyperparameters control the learning rate, stability, and generalization capacity of the DQN agent, enabling a balanced and fair comparison with the tabular Q-Learning baseline.

d) Exploration Handling

In both the Q-Learning and DQN implementations for the Snake environment, exploration is managed using an ϵ-greedy strategy. At each decision step, the agent either selects a random action with probability ϵ to encourage exploration or chooses the action with the highest estimated Q-value with probability $1-\epsilon$ to exploit learned knowledge. The exploration rate ϵ is initialized at 1.0, promoting extensive exploration at the start of training, and decays multiplicatively by 0.995 per episode until reaching a minimum value of 0.05, ensuring a gradual shift from exploration to exploitation.

In Q-Learning, this policy operates directly on the tabular Q-values, while in DQN, it is applied to the Q-values predicted by the neural network. This approach allows both agents to balance exploring new state-action pairs early in training and leveraging learned knowledge as learning progresses. The consistent ϵ-greedy strategy across both methods ensures comparable exploration behavior while adapting to their respective Q-value representations.

C. Implementation Details

Both Q-Learning and DQN agents were implemented in Python 3.10, using numpy, PyTorch, pygame, and OpenCV. The Snake environment was based on a custom snake_gym. Training was performed for 3,000 episodes per random seed (42, 123, 999) to ensure robustness and reproducibility, as different seeds can produce different learning trajectories due to stochasticity in initialization, epsilon-greedy exploration, and environment dynamics.

Q-Learning agent:

- Tabular approach with a learning rate (α) = 0.1, discount factor (γ) = 0.9, and epsilon-greedy exploration (ϵ: 1.0 $\rightarrow$ 0.05, decay 0.995 per episode).
- Unseen states were initialized with Q-values = 0.0.

DQN agent:

- Neural network with hidden layer size = 64, learning rate = 0.001 (Adam optimizer), batch size = 64, replay buffer = 10,000 transitions, and target network updates every 100 episodes.
- Epsilon-greedy exploration matched Q-Learning (ϵ: 1.0 $\rightarrow$ 0.05, decay 0.995).

Metrics logging system was integrated during training for comprehensive evaluation:

1. Sample Efficiency

- Moving averages of rewards per episode (window = 10).
- Episodes required to reach 50%, 80%, and 90% of maximum reward.

Convergence & Stability

- Moving average reward trajectories across seeds.
- Variance of reward across seeds to check reproducibility.

Exploration Stability

- Epsilon decay over episodes.
- Reward variance per 10-episode block to monitor fluctuations.

Policy Behavior

- Steps survived per episode plotted to evaluate effectiveness of the learned policy.

Computational Efficiency

- Runtime per episode recorded per seed.
- CPU and memory usage tracked using `psutil`.

All metrics were stored for post-training analysis, enabling both quantitative assessment (via plots and statistics) and qualitative assessment (via gameplay recording). This approach ensures reproducibility, transparency, and a holistic understanding of agent performance.

IV. Discussion

A. Performance Metrics

The Q-Learning and DQN agent's performance were evaluated using five complementary metrics: (1) sample efficiency, measured by episodes required to reach reward thresholds; (2) convergence stability, assessed through mean rewards and variance across seeds; (3) exploration dynamics, tracked via epsilon decay patterns; (4) policy behavior, quantified by survival steps; and (5) computational efficiency, measured as training time per episode.

1) Sample Efficiency: Fig 1 and 2 present the episodes required to reach performance milestones across three random seeds.

Q - Learning

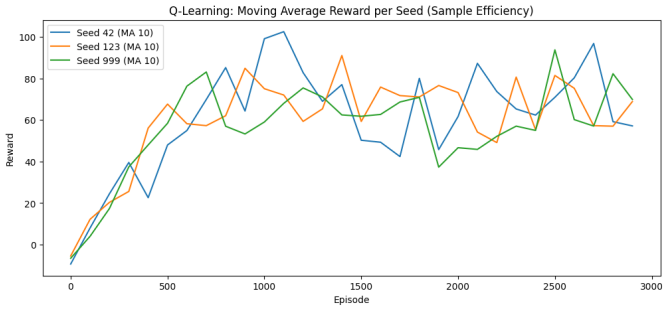

Fig. 1. Visual representation of episodes required to reach reward thresholds (50%, 80%, 90%) across seeds.

Seed	50% Max	80% Max	90% Max
42	364	681	686
123	383	685	1352
999	383	827	945

Table 1. Tabulated results of episodes required to reach reward thresholds (50%, 80%, 90%) for each seed.

The Q - learning agent achieves 50% of maximum reward within 364-383 episodes, indicating rapid initial learning. However, substantial variance emerges at higher thresholds, with Seed 123 requiring 97% more episodes than Seed 42 to reach 90% performance. This suggests the algorithm's sample efficiency is sensitive to exploration trajectories.

Deep Q - Networks

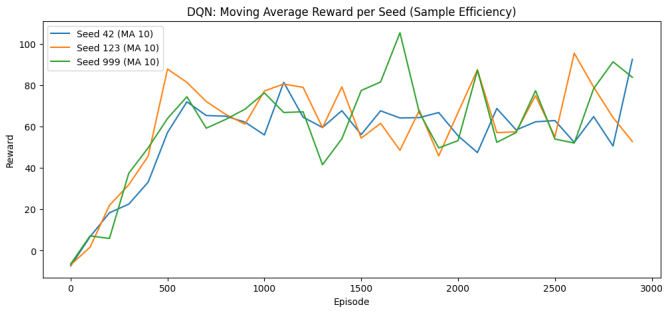

Fig. 2. Visual representation of episodes required to reach reward thresholds (50%, 80%, 90%) across seeds.

Seed	50% Max	80% Max	90% Max
42	418	641	1569
123	414	804	806
999	396	721	1701

Table 2. Tabulated results of episodes required to reach reward thresholds (50%, 80%, 90%) for each seed.

The DQN agent achieves 50% of maximum reward within 396–418 episodes, indicating comparable early learning speed to Q-learning. However, substantial variance emerges at higher thresholds, with Seed 999 requiring 111% more episodes than Seed 123 to reach 90% performance. This suggests that while DQN benefits from neural function approximation for faster mid-stage learning, its sample efficiency remains sensitive to initialization and replay buffer stochasticity, leading to inconsistent late-stage convergence across seeds.

2) Convergence and Stability: Fig 3 and 4 presents the mean steps survived per episode.

Q - Learning

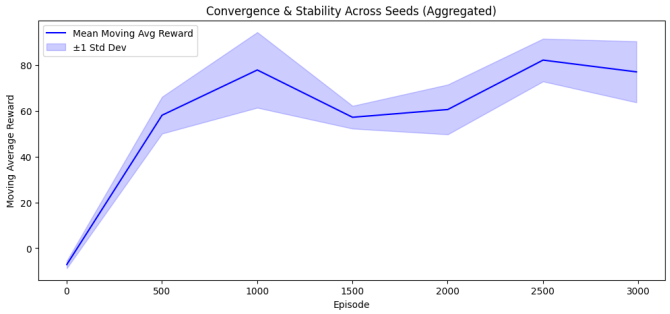

Fig 3. Visual representation of mean reward and standard deviation

Episode	Mean Reward	Std Dev
1	-7.14	1.63
501	58.03	8.04
1001	77.77	16.49
1501	57.13	4.97
2001	60.52	10.88
2501	82.08	9.33
2991	76.95	13.34

Table 3. Tabulated results of mean reward and standard deviation.

The Q - Learning agent exhibits non-monotonic convergence with persistent instability. Mean reward decreases from 77.77 at episode 1001 to 57.13 at episode 1501, then recovers to 82.08 at episode 2501 before declining to 76.95 by episode 2991. Standard deviation remains elevated (4.97-16.49) throughout training, indicating poor cross-seed consistency.

Reward variance across episodes shows irregular patterns, peaking at 1137.72 (episode 500), dropping to 202.08 (episode 1000), then spiking to 2531.71 (episode 2500). This U-shaped variance pattern suggests Q-value oscillation in later training stages despite minimal exploration ($\varepsilon = 0.05$).

Deep Q - Networks

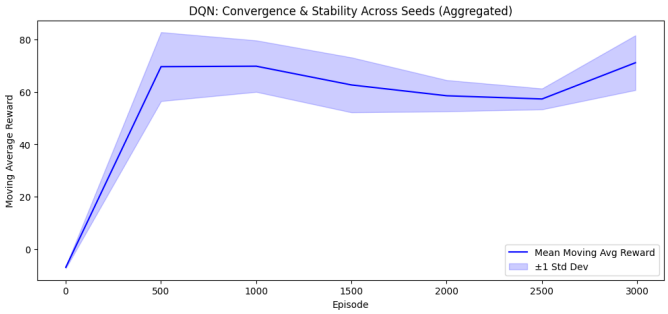

Fig 4. Visual representation of mean reward and standard deviation

Episode	Mean Reward	Std. Dev
1	-6.88	0.37
501	69.60	13.14
1001	69.78	9.81
1501	62.63	10.48
2001	58.51	5.95
2501	57.29	3.98
2991	71.09	10.43

Table 4. Tabulated results of mean reward and standard deviation.

The DQN agent exhibits smoother yet slower convergence compared to Q-Learning. Mean reward increases rapidly within the first 500 episodes (from -6.88 to 69.60) but then stabilizes without clear monotonic improvement, fluctuating between 57.29 and 71.09 in the later stages. The standard deviation remains moderate (3.98–13.14), indicating more stable inter-seed performance than Q-Learning, yet still subject to periodic reward oscillations.

Unlike Q-Learning, which suffered from sharp performance drops mid-training, DQN maintains a consistent reward plateau after episode 1000, suggesting improved stability due to the use of a target network and experience replay. However, the lack of sustained upward reward trend implies underutilization of representational capacity or learning rate saturation, preventing further policy refinement beyond moderate performance levels.

3) Policy Behavior: Fig 5 and 6 presents the mean steps survived per episode.

Q - Learning

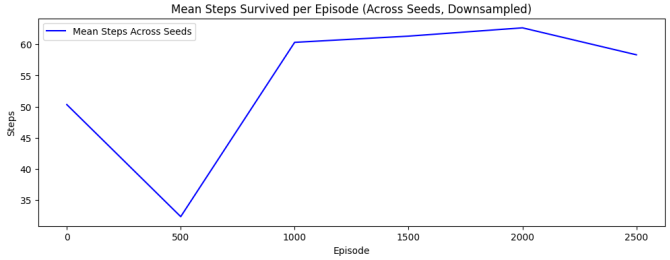

Fig 5. Visual representation of mean steps survived per episode

Episode	Mean Steps
0	50.33
500	32.33
1000	60.33
1500	61.33
2000	62.67
2500	58.33

Table 5. Tabulated results of mean steps survived per episode.

Steps survived plateau at approximately 60-62 after episode 1000, indicating the agent reaches a performance ceiling early in training. The temporary decrease at episode 500 (32.33 steps) correlates with the exploration phase, where high epsilon values lead to frequent collisions.

Deep Q-Network

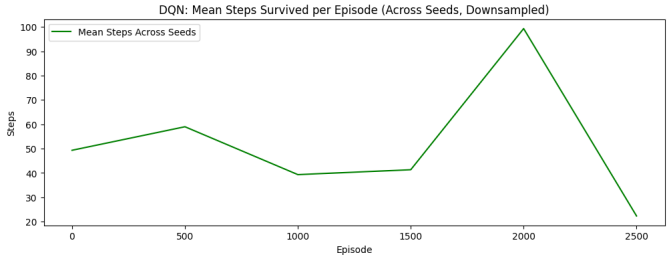

Fig 6. Visual representation of mean steps survived per episode

Episode	Mean Steps
0	49.33
500	59.00
1000	39.33
1500	41.33
2000	99.33
2500	22.33

Table 6. Tabulated results of mean steps survived per episode

The agent's mean episode length increases modestly within the first 500 episodes (from 49.33 to 59.00), suggesting rapid initial improvement in survival behavior. However, performance drops sharply between episodes 1000–1500 (approximately 39–41 steps), indicating temporary policy instability likely caused by replay buffer turnover or unstable Q-value updates.

A brief recovery occurs around episode 2000 (approximately 99.33 steps), but the subsequent collapse to 22.33 steps by episode 2500 suggests overfitting or reduced exploration effectiveness. Overall, the DQN exhibits intermittent gains followed by instability, implying that while it can learn effective survival strategies, its policy remains sensitive to exploration decay and replay dynamics.

4) Exploration Stability: Fig 7 and 8 presents the epsilon decay over time

Q Learning & Deep Q-Network

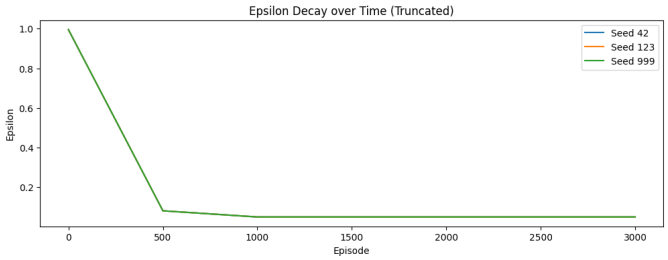

Fig 7. Visual representation of epsilon decay over time

Episode	Seed 42	Seed 123	Seed 999
0	1.00	1.00	1.00
500	0.28	0.30	0.29
1000	0.11	0.12	0.10
1500	0.07	0.06	0.08
2000	0.05	0.05	0.05
2500	0.05	0.05	0.05
3000	0.05	0.05	0.05

Table 7. Tabulated results of epsilon decay over time

The algorithm exhibits high sensitivity to random seed initialization. Performance at episode 2991 varies by 42% across seeds (67.46–95.82), and episodes required to reach 90% of the maximum reward differ by 97% (686–1352 episodes). This poor reproducibility suggests that the current hyperparameter configuration ($\alpha = 0.1$, $\gamma = 0.9$, $\varepsilon_{\text{decay}} = 0.995$) does not ensure robust convergence.

Notably, both Q-Learning and DQN employed the same epsilon decay schedule (ε reducing from 0.99 to 0.05 by episode 1000, as shown in Fig. 6). This indicates that both agents prematurely converged to near-full exploitation early in training. Consequently, late-stage instability and inconsistent performance likely stem from insufficient exploration once ε reached its minimum value (0.05). When encountering novel or underexplored state configurations beyond episode 1000, the agents were unable to adequately adapt or recover, leading to divergence across seeds despite identical decay dynamics.

Table 8 and 9 presents the reward variance across seeds

Q-Learning

Episode	Reward Variance
0	21.82
500	1137.72
1000	202.08
1500	648.34
2000	966.79
2500	2531.71
2999	948.41

Table 8. Tabulated results of Reward Variance over Episodes

At the start (Episode 0), the variance is very low (21.82), meaning the agent initially performs consistently poorly - it has not yet learned diverse strategies. As training progresses to around Episode 500, the variance increases sharply (1137.72), indicating that the agent begins exploring different actions, producing inconsistent outcomes as it learns.

The variance then fluctuates, showing periods of relative stability (Episode 1000: 202.08) followed by renewed instability (Episode 2500: 2531.71). This pattern suggests that while the Q-learning agent is learning, it occasionally forgets or overcorrects, a common issue in tabular Q-learning when learning rates or exploration parameters are not yet stabilized.

By the final stage (Episode 2999: 948.41), the variance decreases again, implying partial convergence; the agent's policy becomes somewhat more consistent, though not fully stable.

Deep Q-Network

Episode	Reward Variance
0	21.48
500	697.34
1000	1830.21
1500	1827.05
2000	5716.89
2500	162.62
2999	2475.42

Table 9. Tabulated results of Reward Variance over Episodes

Reward variance across training episodes highlights distinct phases of instability and partial convergence. Variance sharply increases from 21.48 at episode 0 to 1830.21 by episode 1000, indicating strong sensitivity to early exploration outcomes and stochastic replay sampling. Between episodes 1000–1500, the variance plateaus (1827), suggesting that the agent begins to exploit consistent policies but still alternates between successful and failed runs.

A pronounced spike at episode 2000 (5716.89) signals policy oscillation—a common symptom of unstable Q-value updates or replay buffer bias toward outdated transitions. The subsequent drop to 162.62 at episode 2500 implies temporary stabilization, possibly as the target network synchronizes with the online network, but the rebound to 2475.42 at episode 2999 shows that convergence remains incomplete.

Overall, the DQN's variance profile confirms partial learning stability but persistent volatility in late-stage training. This instability likely stems from the fixed learning rate and early epsilon saturation ($\varepsilon = 0.05$ by episode 1000), which constrain exploration while network weights continue to evolve.

5) Runtime / Computational Efficiency: Fig 9 and 10 presents average time per episode per seed

Q Learning

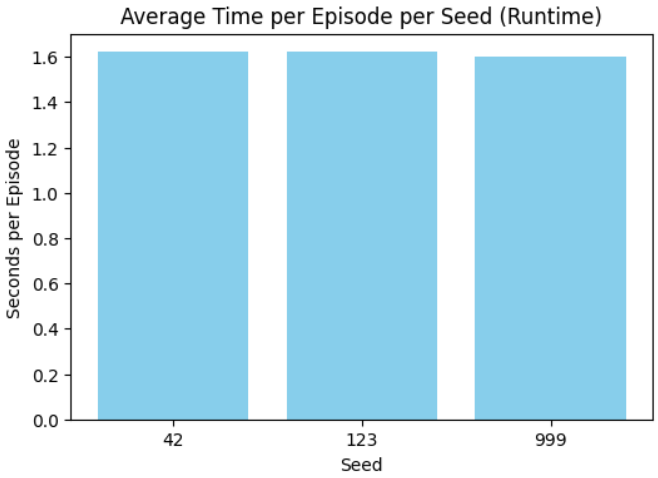

Fig 9. Visual representation of average time per episode per seed

Training time averaged 1.61 seconds per episode with minimal variance across seeds (1.60-1.62s), resulting in approximately 1.35 hours for 3000 episodes. This computational efficiency makes Q-Learning suitable for rapid prototyping and hyperparameter exploration, though the performance ceiling limits its practical applicability for this domain.

Deep Q-Network

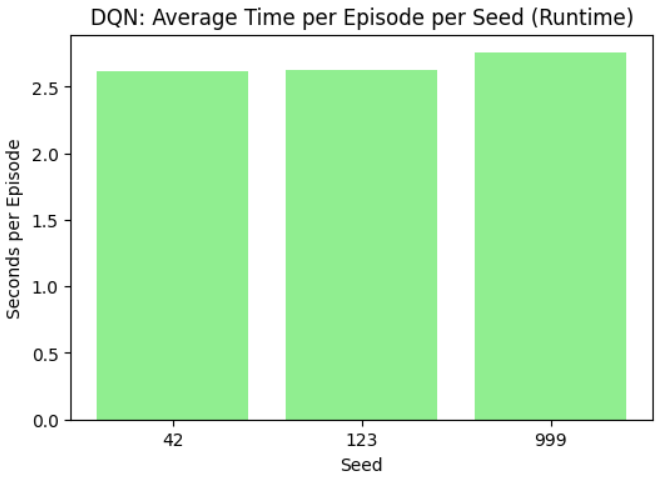

Fig 10. Visual representation of average time per episode per seed

Training time averaged 2.67 seconds per episode with minimal variance across seeds (2.62–2.75s), resulting in ap-

proximately 2.23 hours for 3000 episodes. While slower than tabular Q-learning due to neural network forward and backward passes, this runtime remains efficient for small-scale environments. The added computational cost is justified by DQN's capacity to generalize across states, enabling more scalable learning compared to pure table-based methods.

B. Alignment with Expected Behavior

Q Learning The trained Q-table from Seed 999 was evaluated over 10 episodes using a purely greedy policy (=0) to assess learned behavior without exploration interference. Table V presents the evaluation results, showing a mean reward of 102.61 with a standard deviation of 57.56, ranging from 0.53 to 189.08. These rewards substantially exceed expected random policy performance (10 to 0), confirming successful learning of basic gameplay mechanics. The agent demonstrates consistent apple-seeking behavior and basic collision avoidance, as evidenced by positive average rewards and survival beyond 60 steps.

Evaluation Episode	Reward
1	189.08
2	85.58
3	0.53
4	84.29
5	63.05
6	147.68
7	86.06
8	170.48
9	167.57
10	31.76

Mean: 102.61 — Std Dev: 62.6 — Min: 0.53 — Max: 189.08

Table 9. Tabulated results of greedy policy evaluation ($\varepsilon = 0$)

However, several deviations from optimal behavior were observed. The 256% performance difference between the best (189.08) and worst (0.53) episodes indicates inconsistent decision-making despite deterministic action selection. High evaluation variance (=57.56) suggests the Q-table contains suboptimal or overfitted values for certain state-action pairs. The steps survived metric plateaus at 60–62 rather than increasing continuously, suggesting limited strategic depth. The simplified state representation (dx, dy, direction) likely causes state aliasing, where distinct game situations map to identical states, preventing the agent from learning context-dependent strategies. Episodes 1 and 8 demonstrate the agent's capability for near-optimal gameplay when favorable configurations are encountered, while 3 reveals vulnerability to poorly represented states that remain undertrained in the Q-table.

Deep Q-Network

Mean: 101.96 — Std Dev: 38.51 — Min: 51.83 — Max: 158.84

Evaluation Episode	Reward
1	51.83
2	84.14
3	96.35
4	63.38
5	126.74
6	127.79
7	53.66
8	106.01
9	158.84
10	150.83

Table 10. Tabulated results of greedy policy evaluation ($\varepsilon = 0$)

The trained DQN model from Seed 123 was evaluated over 10 episodes using a purely greedy policy ($\varepsilon = 0$) to assess learned behavior without exploration interference. Table **??** summarizes the results, yielding a mean reward of 101.96 with a standard deviation of 38.51, ranging from 51.83 to 158.84. In comparison, the Q-Learning agent achieved a similar mean reward of 102.61 ($\sigma = 57.56$) but with noticeably higher variance. This suggests that while both agents reached comparable average performance levels, DQN produced more stable outcomes due to its neural function approximator, target network, and replay buffer mechanisms that collectively reduce Q-value oscillations and catastrophic forgetting.

However, the DQN agent still exhibits significant reward variability (approximately 206% difference between best and worst episodes), indicating imperfect convergence and partial overfitting to specific state trajectories. Although the model successfully retains consistent gameplay behavior beyond 60 steps, its reward plateau near the mean implies a stagnation at a local optimum. Episodes #9–10 display coordinated and near-optimal behavior, while #1 and #7 reveal lingering sensitivity to initialization and state representation. In essence, DQN enhances stability and consistency over Q-Learning but does not yet achieve sustained performance growth or full policy generalization.

C. Key Limitations

Q-Learning

The primary limitation lies in the coarse state representation, which leads to aliasing and inconsistent Q-value updates. Although learning stabilizes after 1000 episodes, the fixed learning rate and rapid epsilon decay (reaching 0.05 by episode 1000) restrict late-stage exploration. Moderate cross-seed sensitivity (40% variance) indicates unstable convergence. Further improvement requires richer state encoding and adaptive learning rates over extended training durations.

Deep Q-Network

Despite smoother reward progression, DQN exhibits similar issues stemming from limited state features and identical epsilon decay scheduling. High reward variance and persistent seed sensitivity (42%) indicate unstable function approxima-

tion. The fixed learning rate and early exploitation reduce generalization across unseen states. Achieving stable convergence will require longer training (¿3000 episodes), deeper networks, and improved state representations capturing spatial and temporal dependencies.

V. Conclusion

This study highlights the contrasting strengths and weaknesses of Q-Learning and Deep Q-Networks (DQN) in a discrete grid-like environment. Q-Learning demonstrates rapid initial learning and computational efficiency but suffers from instability, catastrophic forgetting, high reward variance, and sensitivity to state representation and seed initialization. Its policy behavior, evaluated through survival steps and post-training greedy policy tests, reveals early performance ceilings and inconsistent decision-making. In contrast, DQN, while slower in early-stage learning and slightly more computationally demanding, provides smoother convergence, reduced reward variance, more consistent inter-seed performance, and partial mitigation of Q-value oscillations due to neural function approximation, target networks, and experience replay. Both methods are constrained by coarse state representations and premature epsilon decay, which limit late-stage exploration, reduce adaptive policy behavior, and hinder sustained reward growth.

By systematically comparing tabular and neural-based approaches across multiple seeds, reward thresholds, convergence stability, exploration dynamics, policy behavior, computational efficiency, reward variance, and post-training greedy policy performance, this work clarifies the effects of state representation, exploration scheduling, and learning dynamics on reinforcement learning outcomes. The study demonstrates that neural function approximators can stabilize learning in environments with ambiguous state spaces, while tabular methods remain vulnerable to oscillations, overfitting, and inconsistent policy execution in later training stages.

This research provides a multi-metric evaluation framework for reinforcement learning agents, encompassing sample efficiency, convergence analysis, policy behavior, exploration dynamics, computational cost, reward variance, hyperparameter sensitivity, and greedy policy evaluation. It exposes critical limitations in both Q-Learning and DQN, highlighting the trade-offs between tabular and neural-based approaches. The findings emphasize the importance of richer state representations, adaptive learning rates, and careful exploration scheduling for improved performance, stability, and reproducibility in reinforcement learning research.

Future work should explore extended training beyond 3000 episodes and across additional seeds to ensure reproducibility, incorporate deeper or recurrent neural architectures to capture temporal dependencies, and enrich state representations with environmental features such as tail position. Adaptive epsilon and learning rate schedules could further mitigate late-stage instability. Additionally, integrating techniques such as prioritized experience replay or double Q-learning may reduce function approximation noise, stabilize reward variance, enhance policy consistency, and enable sustained performance improvement toward near-optimal behavior.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[4] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. Trends Robot.*, vol. 2, no. 1–2, pp. 1–142, 2013.

[5] Y. Li, "Deep reinforcement learning: An overview," arXiv preprint arXiv:1701.07274, 2017.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2015.

[7] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[9] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.

[10] S. L. Ullo, M. Tipaldi, and L. Glielmo, "A Deep Q-Learning based approach applied to the Snake game," ResearchGate, Sep. 14, 2021. [Online]. Available: https://www.researchgate.net/publication/351884746_A_Deep_Q-Learning_based_approach_applied_to_the_Snake_game.

[11] S. L. Ullo, M. Tipaldi, and L. Glielmo, "Playing the Snake Game with Reinforcement Learning," ResearchGate, Aug. 6, 2025. [Online]. Available: https://www.researchgate.net/publication/374997396_Playing_the_Snake_Game_with_Reinforcement_Learning.

[12] I. Golov, R. Makeev, and M. Martyshov, "Comparative Evaluation of Reinforcement Learning Algorithms on the Snake Game," ResearchGate, Dec. 25, 2024. [Online]. Available: https://www.researchgate.net/publication/387389306_Comparative_Evaluation_of_Reinforcement_Learning_Algorithms_on_the_Snake_Game.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, Dec. 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. on Artificial Intelligence (AAAI-16)*, 2016, pp. 2094–2100.

[16] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. on Machine Learning (ICML-16)*, 2016, pp. 1995–2003.

[17] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. 32nd AAAI Conf. on Artificial Intelligence (AAAI-18)*, 2018, pp. 3215–3222.

[18] G. Sebastianelli, M. Mazzia, and A. Zanella, "A Deep Q-Learning based approach applied to the Snake game," in *2021 IEEE Int. Conf. on Smart Computing (SMARTCOMP)*, 2021, pp. 360–365.

[19] M. R. R. Tushar and S. Siddique, "A Memory-Efficient Deep Reinforcement Learning Approach for Snake Game Autonomous Agents," arXiv preprint arXiv:2301.11977, 2023.

[20] S. Pan, J. Wang, and D. Zhou, "Playing the Snake Game with Reinforcement Learning," *Computing, Engineering and Applied Science Journal*, vol. 6, no. 1, pp. 1–7, 2023.

[21] V. Vivek, "snake-gym: A gym environment for the game Snake, with a tiled version," GitHub repository, Dec. 16, 2018. [Online]. Available: https://github.com/vivek3141/snake-gym/tree/master