Deep Q-Snake: An Intelligent Agent Mastering the Snake Game with Deep Reinforcement Learning

Debjyoti Ray
Department of ECE
IIIT - Allahabad
Prayagraj, India
iec2022111@iiita.ac.in

Arindam Ghosh
Department of IT
IIIT - Allahabad
Prayagraj, India
rsi2021001@iiita.ac.in

Muneendra Ojha
Department of IT
IIIT - Allahabad
Prayagraj, India
muneendra@iiita.ac.in

Krishna Pratap Singh
Department of IT
IIIT - Allahabad
Prayagraj, India
kpsingh@iiita.ac.in

TENCON 2024 - 2024 IEEE Region 10 Conference (TENCON) | 979-8-3503-5082-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/TENCON61640.2024.10903025

***Abstract*—The Deep Q-Network (DQN) has emerged as a robust deep reinforcement learning algorithm capable of learning optimal policies in complex, high-dimensional environments. The Snake game presents unique challenges to an agent's training due to its dynamic and unpredictable nature, making it an intriguing test bed for reinforcement learning techniques. In this paper, we explore the development of "Deep Q-Snake," an intelligent agent that leverages the power of DQN to master the classic Snake game. By employing DQN, Deep Q-Snake learns to navigate the game grid, collect food, grow in length, and avoid self-collisions and obstacles. Extensive training and experimentation demonstrate Deep Q-Snake's impressive proficiency in playing the Snake game, achieving high scores, and showcasing strategic gameplay. This work exemplifies DQN's potential to enable intelligent agents to excel in challenging environments and surpass human-level performance when playing the Snake game.**

***Index Terms*—Snake Game, Deep Q-Networks, Q-Learning, Markov Decision Process, Deep Reinforcement Learning, Pygame**

I. Introduction

In recent years, the field of artificial intelligence has made impressive progress, particularly in the area of deep reinforcement learning (DRL). DRL has proven to be an exceptional tool for training intelligent agents to excel in complex and dynamic environments. These agents, which are based on sophisticated algorithms, showcase remarkable capabilities in a wide range of tasks, such as robotic control and strategic decision-making. The classic Snake game is a popular and intriguing testbed [1] used to evaluate intelligent agents' strategic decision-making and spatial awareness skills through the DRL framework.

The Deep Q-Network (DQN) algorithm, a groundbreaking development in deep reinforcement learning, has revolutionized how intelligent agents learn to navigate and make decisions in complex environments. It was introduced by Volodymyr Mnih in their seminal paper in 2015 [2]. The DQN algorithm leverages deep neural networks to approximate the optimal action-value function, enabling agents to learn effective strategies directly from input data. In the original paper, the DQN algorithm demonstrated its capabilities by training agents to play a range of classic Atari 2600 games directly from pixel input, outperforming previous approaches and achieving human-level performance on multiple games.

A novel DQN-based agent was developed in [3] to improve deep reinforcement learning in real-time Atari games. It combines planning-based approaches and model-free DQN agents to achieve higher scores and better efficiency in dynamic environments. A study [4] on applying DQN to a visual fighting game with two players and 11 optimized actions shows promising results. Control parameters were systematically tested on a visual platform, highlighting DQN's potential effectiveness in real-time fighting games. The effectiveness of the DQN algorithm and other deep reinforcement learning techniques has also been investigated in [5] for video games using OpenAI Gym's Arcade Learning Environment to apply advanced reinforcement learning to a wide range of games.

Our primary objective in this paper is to showcase the potential of Deep Q-Snake in mastering the Snake game through the application of DQN. Moreover, we analyze the performance of the DQN algorithm with different hyperparameter settings. The contributions of the research are as follows:

- Design the Snake Game Environment and train an agent through the baseline DQN algorithm, which is capable of playing the game.
- Analyze and compare the performance of the DQN algorithm with different hyperparameter settings like different learning rates, batch sizes, rates of epsilon decay, etc.

Deep Q-Snake showcases how reinforcement learning can create intelligent agents capable of making strategic decisions in complex gaming environments. The remainder of this paper is organized as follows: Section II provides an overview of related work. Section III describes the methodology, detailing the problem statement, the Snake game environment, the DQN architecture, and the training process for Deep Q-Snake. In Section IV, we present and analyze the experimental results, showcasing the agent's performance. Finally, Section V concludes the discussion with our findings and future research directions.

II. Related Work

Reinforcement learning (RL) has emerged as a valuable approach, enabling agents to learn from their environment. It finds applications in various domains, including manufacturing, inventory and delivery management, power systems, finance, and self-driving cars. [6] explores implementing reinforcement learning to create an artificial agent capable of playing games. Specifically, in the realm of training agents to play the Snake game, the utilization of Deep Q-Networks (DQN) has been a noteworthy endeavor in prior research. Various studies have successfully employed DQN to enable agents to navigate and make decisions within the dynamic environment of the Snake game. In [7], a solution to the challenge of developing AI systems that can learn to play classic games is presented. It proposes using Deep Q-Learning for playing Snake and provides numerical simulations for both the training and testing phases.

The effect of different learning rates of the SARSA algorithm is analyzed over several iterations of the agent playing the Snake game in [8]. Authors in [1] address the challenges in applying deep reinforcement learning (DRL) in complex scenarios with sparse and delayed reward signals. A DRL model using a convolutional neural network (CNN) trained with a Q-learning variant and a carefully designed reward mechanism is employed to train the agent in the Snake game. [9] explores the application of reinforcement learning techniques, specifically Q-Learning, SARSA, and Proximal Policy Optimization (PPO), to the Snake video game. Q-Learning and SARSA fell short of generating optimal results due to the game's expansive environment. PPO was implemented with different input approaches, and the raycasting method demonstrated the best performance, showcasing the snake agent's proficiency in both collecting food and avoiding obstacles.

Reinforcement learning proves effective when multiple decisions are needed to solve a problem. [10] showcases the outcomes of employing a DQN to play the Snake game. However, an improved reward setting and augmented snake's awareness of its surroundings result in higher scores and mitigate overfitting. Additionally, the distinctions between DQN and traditional evolutionary algorithms and potential optimizations for DQN are explored.

The challenge of resource-intensive requirements in DRL methods is addressed in [11]. Many applications prioritize minimizing memory usage and computational time over achieving maximum rewards. A lightweight convolutional neural network with a variant of the Q-network is employed to demonstrate reasonable performance with compressed imagery data. This approach eliminates the need for additional environmental information and reduces memory and time requirements.

It is evident that reinforcement learning has made significant progress in gaming platforms, especially when integrated with neural networks. This approach has become increasingly popular as DRL, enabling agents to master complex tasks.

III. Methodology

In this section, we delve into the intricate web of methodologies employed to imbue our Snake game agent with the prowess of DRL. Also, the fundamentals of Q learning and DQN algorithm, Snake game development and experimental setup have been discussed.

A. Problem Statement

This section explores the comprehensive methodologies utilized to enhance our Snake game agent with deep reinforcement learning (DRL) capabilities. Additionally, we discuss the fundamentals of Q-learning and the Deep Q-Network (DQN) algorithm while developing the Snake game and the experimental setup.

State Space: The state space represents all possible configurations the Snake game can take at a given point of time. In the Snake game, the state includes information about the snake's direction, the food's location, and the game grid's wall layout. The state space S can be defined as a set 12 binary digits: $S = \{s_1, s_2, ..., s_{12}\}$, where s_i is a specific state information within the space. It consists of:

- s_1 : Food Up
- s_2 : Food Down
- s_3 : Food Left
- s_4 : Food Right
- s_5 : Snake Direction Up
- s_6 : Snake Direction Down
- s_7 : Snake Direction Left
- s_8 : Snake Direction Right
- s_9 : Wall direction Up
- s_{10}: Wall direction Down
- s_{11}: Wall direction Left
- s_{12}: Wall direction Right

Action Space: The action space comprises all possible moves or decisions that the agent can make in a given state. Typical actions include moving the snake up, down, left, or right. The action space A can be defined as: $A = \{a_1, a_2, a_3, \ldots, a_m\}$, where a_i represents a specific action. In the Snake game, the action space is discrete, consisting of the set $\{Up, Down, Left, Right\}$. The agent selects an action from this set based on its policy learned through the learning process. If the snake is currently moving up, the valid actions would be $\{Left, Right, Up\}$, while $\{down\}$ would not be a valid action at that moment. Similarly, if the snake is moving $left$, the valid actions would be $\{Up, Down, Left\}$, while $\{right\}$ would be excluded. This ensures that the actions are consistent with the game mechanics and prevents the snake from immediately reversing its direction, leading to a more realistic and feasible action space for the Snake game.

B. Fundamentals of Q learning

Q-learning, a reinforcement learning technique, is employed to identify the optimal policy for action selection in a given finite Markov Decision Process (MDP). This algorithm is especially adept at addressing scenarios where an agent makes sequential decisions within an environment [9].

Markov Decision Process (MDP): An MDP is a mathematical framework for modeling sequential decision-making under uncertainty. It exhibits the Markov property, wherein the system's subsequent state relies solely on the present state and action, independent of the sequence of prior events [12]. It is represented as (S, A, P, R, γ), where S is a set of states representing all possible situations in the environment, A is a set of actions that the agent can take in each state, a transition model P is the probability distribution of transitioning from one state to another given a specific action and it is represented as$(P(s'|s,a)$, R is the immediate reward received by the agent, and $\gamma(0 < \gamma < 1)$ is the discount factor [13].

The core concept of Q learning is the Bellman Equation, which embodies the principle of optimality. It expresses the value of a state as the sum of the immediate reward and the expected value of the subsequent state, factoring in the optimal action selection. Mathematically, it is represented as:

$$V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')] \tag{1}$$

C. Deep Q Network Algorithm

DQN revolutionizes RL by leveraging neural networks to approximate Q-values in large, continuous state spaces. DQN extends traditional Q-learning by using a neural network to approximate the Q-function $(Q(s,a))$ instead of maintaining a Q-table [2]. This allows DQN to handle complex state representations. The DQN framework is grounded in two fundamental components:

- **Target Q-Networks**: DQN uses a target Q-network to stabilize training. This is a separate neural network with the same architecture as the main network, but its parameters are updated less frequently. The target network helps mitigate the issues of non-stationarity during learning.
- **Loss Function**: DQN employs the mean squared temporal difference (TD) error as its loss function, as expressed in equation 2. This TD error represents the disparity between the predicted Q-value and the target Q-value. The latter is determined by adding the immediate reward to the discounted maximum Q-value of the subsequent state.

$$\delta = (Q_{target}(s,a;\theta') - Q(s,a;\theta))^2 \tag{2}$$

The primary neural network within the framework of DQN serves to approximate the Q-function, symbolized as $Q(s,a;\theta)$, where θ denotes the parameters specific to the main neural network. The target Q-value is derived by summing the immediate reward (r) with the discounted maximum Q-value of the subsequent state (s') obtained from the target Q-network. This is denoted as $Q_{\text{target}}(s,a;\theta')$, with θ' representing the parameters specific to the target Q-network.

The training of DQN entails a repetitive process of engaging with the environment, gathering experiences, and adjusting the neural network parameters to minimize the temporal difference (TD) error. The structure of the DQN is illustrated in the diagram provided in Figure 1. DQN commonly utilizes an epsilon-greedy approach to manage exploration and exploitation trade-offs. The agent, with a chance of ϵ, opts for a random action, while with a probability of $1-\epsilon$, it selects the action associated with the highest Q-value. DQN introduces mechanisms like experience replay and target networks to enhance stability during training and improve convergence.

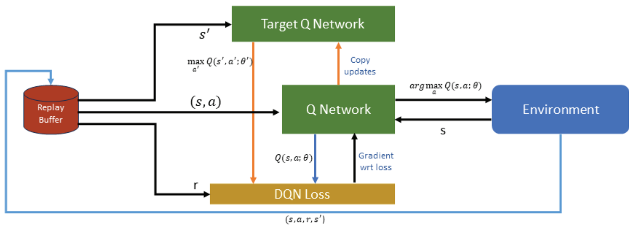

Fig. 1. DQN Structure

D. Snake Game Development using Pygame

The Snake Game, a classic arcade game originating in the late 1970s, is known for its simple yet addictive gameplay. In the context of mobile gaming, Snake Games have been adapted for smartphones and tablets, often available as mobile applications [14]. For this study, a similar Snake game environment was developed using the Pygame library on the Python platform. Figure 2 illustrates the designed Snake game environment. Pygame, a popular library for creating 2D games in Python, was utilized to depict the snake as a series of connected black segments, with the food represented by a green square.

In the game, the snake's position and length are dynamically adjusted based on the actions executed by the Agent. The food is randomly generated on the screen, ensuring it does not overlap with the snake. Figure 3 illustrates the user interface depicting the game's progress. A collision detection mechanism is implemented to determine if the snake interacts with itself or consumes the food. The game terminates when the snake collides with itself or reaches the screen boundaries. The snake, food, and other game elements are rendered on the screen through a rendering function, with the screen being continuously updated in each iteration of the game loop.

E. Experimental Setup and Hyperparameters

The configuration of experiments and the assignment of hyperparameter values are critical components of any research work. For the Snake Game, the experimental setup includes detailed configurations such as the DQN architecture and Q-table.

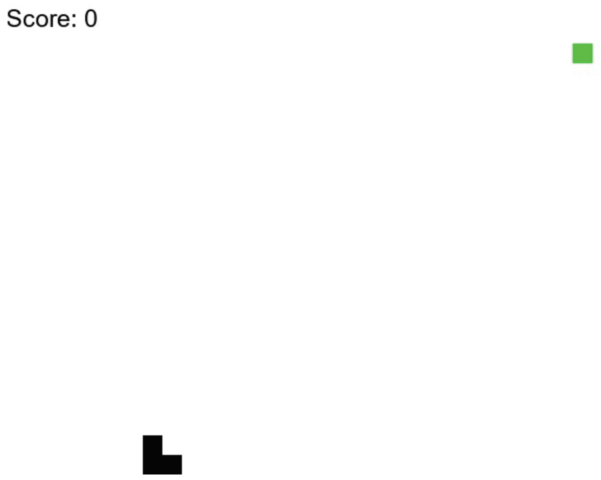

Fig. 2. Snake Game Environment

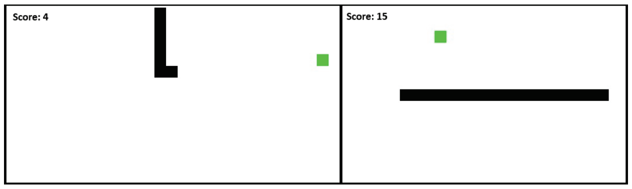

Fig. 3. Snake Game Progress

DQN Architecture: The DQN architecture is implemented using a Sequential model in the Keras library. The model comprises three densely connected layers. The first layer accepts the input shape of the environment, represented by the state dimension, and applies the ReLU activation function to introduce non-linearity. This is followed by another dense layer with a ReLU activation function. The final layer consists of dense neurons corresponding to the possible actions in the environment. The model is compiled using the mean squared error (MSE) loss function and the Adam optimizer with a specified learning rate.

In the context of model training, hyperparameters refer to the settings and configurations that govern the learning process. These parameters are external to the model and are tuned to optimize the training process. Table I represents the values of hyperparameters that are used in the experiment.

IV. Result and Discussion

This section presents and analyzes the experimental outcomes, examining performance metrics and the behavior of the trained model to derive insights. The discussion aims to provide a comprehensive understanding of the implications and potential improvements for the Snake Game.

TABLE I
HYPERPARAMETERS

Algorithm	Hyperparameters	Values
DQN	Learning Rate	0.1
		0.01
		0.001
	Batch Size	64
		128
		256
	ϵ maximum	1.0
	ϵ decay	0.1
		0.5
		0.9
	γ	0.9
Q Learning	Learning Rate	0.001
	ϵ	0
	γ	0.9

A. Q Learning

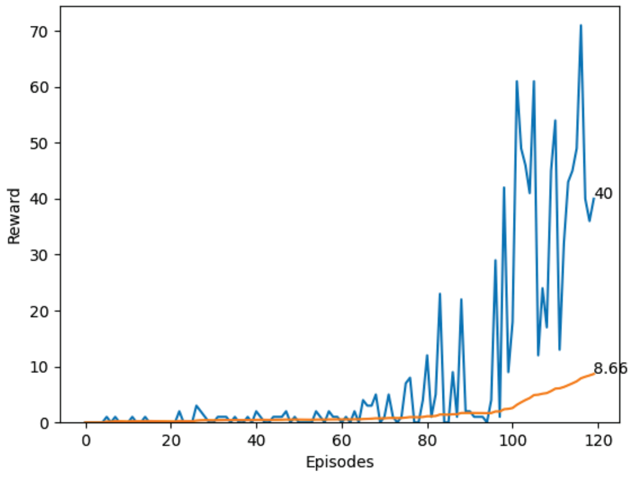

Fig. 4. Performance Graph of Q-learning Algorithm

While demonstrating potential, the application of Q-learning to the Snake game reveals suboptimal learning efficiency in the current experimental setup. After 120 episodes of training, the maximum cumulative reward achieved is 70, indicating a limited mastery of the game environment. The average score over the training period stands at 8.6, suggesting challenges in consistently achieving high performance. Furthermore, each iteration of the game comprises 50 steps, providing a context for the agent's decision-making within a finite time frame.

Figure 4 illustrates the performance graph across the number of episodes. The graph features two line plots: the blue line denotes the cumulative reward, while the orange line indicates the average payment. The graph illustrates that the cumulative reward escalates with the rise in the number of occurrences. The Q-learning method fails to attain substantial cumulative and average rewards for the Snake game within a restricted number of episodes.

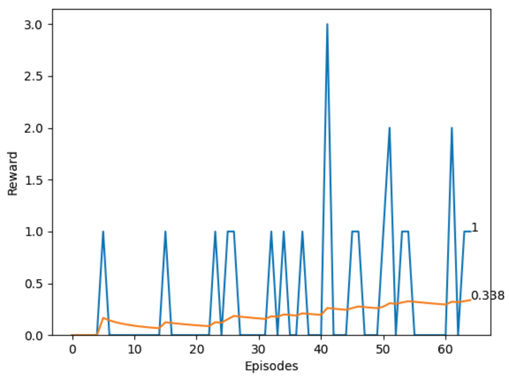

Fig. 5. Reward vs Episode graph in Q-learning

The rewards gained over the number of episodes are illustrated in Figure 5. The graph has two line plots: the blue plot is for actual rewards, and the orange plot is for average rewards. The blue line has a higher peak at around episode 30, and the orange line has a lower peak at around episode 60.

B. DQN

The application of DQN to the Snake game reveals a moderate learning efficiency in the current setup. After 100 episodes of training, the maximum cumulative reward achieved is approximately 450, suggesting a reasonable understanding of the game dynamics. Each iteration of the game consists of 50 steps, providing a temporal context for the agent's decision-making. Figure 6 depicts the performance graph for three different learning rates over 100 episodes.

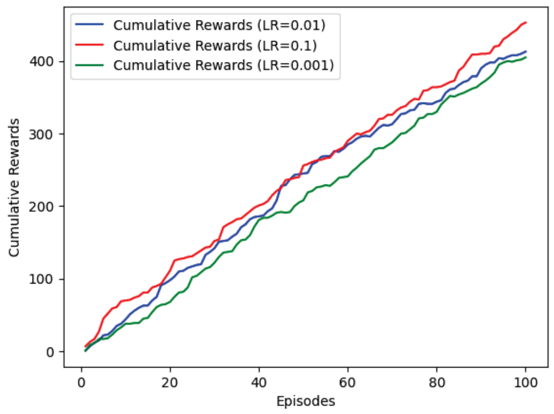

Fig. 6. Performance Graph of DQN Algorithm

The learning efficiency was evaluated across three different learning rates: 0.1, 0.01, and 0.001. The cumulative reward trends for these learning rates demonstrate varying degrees of performance. The choice of learning rate significantly impacts the agent's ability to adapt to the dynamic environment, affecting overall training effectiveness. To assess the performance of the DQN, a reward graph is presented in Figure 7, illustrating the actual rewards obtained during training. This graph includes three distinct learning rates—0.1, 0.01, and 0.001—allowing for an analysis of how variations in learning rates influence the DQN's performance in the Snake game. The rewards range from -20 to 20, with the blue plot showing the highest cumulative rewards and the green plot displaying the lowest overall rewards.

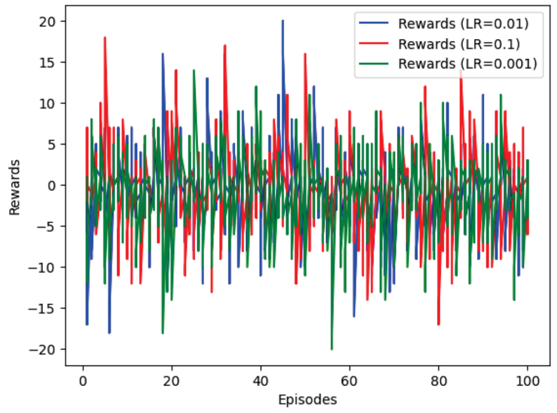

Fig. 7. Reward vs Episode graph in DQN

To evaluate the efficacy of the DQN, a reward graph corresponding to three different batch sizes—64, 128, and 256—is presented in Figure 8. These graphs visually depict the actual rewards attained by the agent throughout the training episodes, with rewards fluctuating between -20 and 20. This observation suggests that, in the context of our Snake game implementation, the learning process may not be notably sensitive to adjustments in batch size.

In the context of DQN, the epsilon-greedy strategy is commonly employed. Initially, the agent utilizes a higher exploration rate (ϵ) to encourage state space exploration and discover optimal actions. As training progresses, ϵ gradually decreases, favoring the exploitation of learned knowledge to maximize rewards. The impact of ϵ on the cumulative rewards of the agent over the episodes is shown in Figure 9. Three different decay values, 0.01, 0.1, and 0.9, are used in the experiment, with the decay value of 0.01 taking more time to achieve experience and collect substantial rewards.

Compared to conventional Q-learning, our findings demonstrate the effectiveness of the Deep Q-Network (DQN) within the scope of our Snake Game project. We assess and compare the metrics, emphasizing the benefits and possible improvements the DQN methodology provides in contrast to the traditional Q-learning approach.

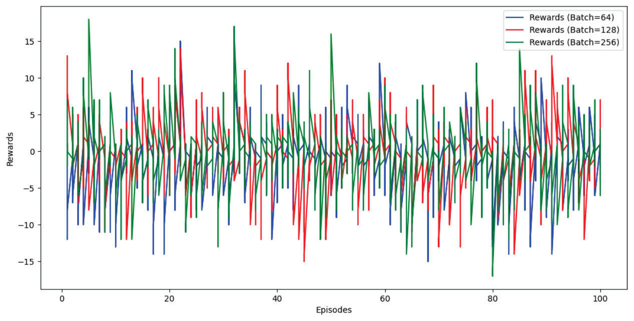

Fig. 8. Reward vs Episode graph for different Batch Size in DQN

Conclusion and Future Work

This study demonstrates that Deep Q-Learning enhances the gameplay intelligence of the Snake Game. Through the meticulous design of the DQN architecture and deliberate hyperparameter tweaking, the agent's navigation and decision-making abilities were markedly enhanced inside the dynamic gaming environment. The trained model demonstrated impressive proficiency in the Snake Game, exhibiting adaptability in many settings and providing valuable insights for the broader application of deep reinforcement learning in gaming contexts.

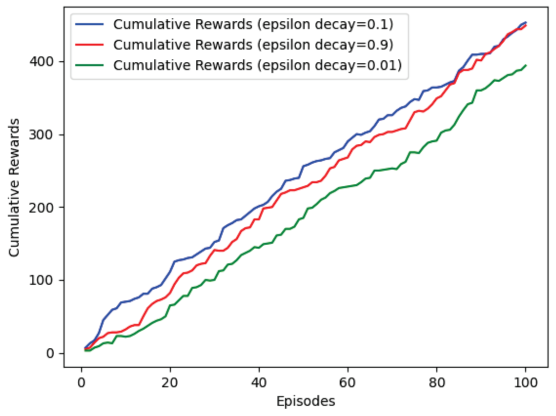

Fig. 9. Reward vs Episode graph for different Batch Size in DQN

This research's insights extend beyond gaming, offering valuable lessons for intelligent decision-making in dynamic real-world contexts. The integration of DRL with traditional arcade games enhances academic discussion and exemplifies the advancing domain of AI. The Snake Game, once a nostalgic activity, now serves as a platform for exploring the limits of attainable objectives, demonstrating the transformative potential of reinforcement learning in developing intelligent entities.

Future studies in this field may explore the scalability of the suggested DQN architecture to more intricate gaming contexts to get insights into policy generalization. Furthermore, incorporating sophisticated methodologies such as curriculum learning or transfer learning may improve the agent's flexibility. Investigating the interpretability of acquired policies and comprehending decision-making processes enhances explainable AI. Deep Reinforcement Learning (DRL) extends beyond gaming applications to address real-world challenges like robots and autonomous systems, presenting a promising direction for future research.

References

[1] Z. Wei, D. Wang, M. Zhang, A.-H. Tan, C. Miao, and Y. Zhou, "Autonomous agents in snake game via deep reinforcement learning," in *2018 IEEE International Conference on Agents (ICA)*, 2018, pp. 20–25.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," *Advances in neural information processing systems*, vol. 27, 2014.

[4] S. Yoon and K.-J. Kim, "Deep q networks for visual fighting game ai," in *2017 IEEE conference on computational intelligence and games (CIG)*. IEEE, 2017, pp. 306–308.

[5] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game ai," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.

[6] L. Narang and A. Tickoo, "Implementing reinforcement learning to design a game bot," in *Applications of Artificial Intelligence and Machine Learning: Select Proceedings of ICAAAIML 2021*. Springer, 2022, pp. 287–302.

[7] A. Sebastianelli, M. Tipaldi, S. L. Ullo, and L. Glielmo, "A deep q-learning based approach applied to the snake game," in *2021 29th Mediterranean Conference on Control and Automation (MED)*. IEEE, 2021, pp. 348–353.

[8] A. J. Almalki and P. Wocjan, "Exploration of reinforcement learning to play snake game," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 377–381.

[9] R. S. Bonnici, C. Saliba, G. E. Caligari, and M. Bugeja, "Exploring reinforcement learning: A case study applied to the popular snake game," in *Disruptive Technologies in Media, Arts and Design*, A. Dingli, A. Pfeiffer, A. Serada, M. Bugeja, and S. Bezzina, Eds. Cham: Springer International Publishing, 2022, pp. 169–192.

[10] P. Yuhang, S. Yiqi, M. Qianli, G. Bowen, D. Junyi, and T. Zijun, "Playing the snake game with reinforcement learning," *Cambridge Explorations in Arts and Sciences*, vol. 1, no. 1, 2023.

[11] M. R. R. Tushar and S. Siddique, "A memory efficient deep reinforcement learning approach for snake game autonomous agents," in *2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, 2022, pp. 1–6.

[12] D. P. Bertsekas, "Feature-based aggregation and deep reinforcement learning: a survey and some new implementations," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 1, pp. 1–31, 2019.

[13] R. S. Sutton and A. G. Barto, *"Finite Markov Decision Processes" in Reinforcement learning: An introduction*. MIT press, 2018.

[14] G. Bateson, "The history of snake the game," 2023, accessed on September 30, 2024. [Online]. Available: https://www.coolmathgames.com/blog/the-history-of-snake-the-game