Deep Q-Network (DQN) Implementation for Playing Snake Game

Madhav Badewad
IIITA (Btech-ECE)
IEC2020030
iec2020030@iiita.ac.in

Saumya Jain
IIITA (Btech-ECE)
IEC2020088
iec2020088@iiita.ac.in

Harsh Raj
IIITA (Btech-ECE)
IEC2020131
iec2020131@iiita.ac.in

***Abstract*—This paper presents the implementation of a Deep Q-Network (DQN) for playing the classic Snake game using the Turtle graphics library. The objective of this study is to demonstrate the effectiveness of the DQN algorithm in learning and improving gameplay strategies for the Snake game.**

The methodology involves training an agent to play the Snake game by employing the DQN algorithm. The game environment is created using the Turtle graphics library in Python, providing a visually interactive interface. The DQN is trained using a combination of experience replay and target network techniques to stabilize and enhance learning. The agent receives state information from the game, such as the position of the snake, its length, and the location of the food, and takes actions accordingly. Results show that the trained DQN agent achieves impressive performance in playing the Snake game. Through the learning process, the agent successfully learns to navigate the game board, avoid obstacles, and consume food strategically to maximize its score. The agent demonstrates an ability to adapt and learn from previous experiences, resulting in continuous improvement over time.

The implementation highlights the potential of DQN algorithms in solving complex tasks, even in simple game environments like Snake. The use of Turtle graphics facilitates visualization and enhances the interactive nature of the Snake game. The findings of this study contribute to the understanding of reinforcement learning techniques and their applicability in game-playing scenarios.

***Index Terms*—Deep Q-Network (DQN), Snake game, reinforcement learning, Turtle graphics, Python.**

I. Introduction

The snake game has a timeless appeal, enthralling players with its simple yet intriguing action that has lasted the test of time. Players must manoeuvre a snake through a grid, consuming food to expand its length while avoiding collisions with its own body or the grid's boundaries. The game's allure stems from the mix of strategic decision-making, reflexes, and spatial awareness necessary to navigate the game's dynamic and ever-changing world.

For decades, players of all ages have been captivated to the addictive appeal and simple principles of the snake game. The aim of guiding the snake to eat food while avoiding obstacles demands players to prepare ahead of time, organise their motions, and analyse the repercussions of each action. This combination of ability and insight requires players to constantly alter their tactics as the game develops.

The game's natural popularity can be traced to its ease of use and intuitiveness. The simple rules and goals are immediately grasped by players, allowing for fast involvement and enjoyment. Mastering the game, on the other hand, necessitates improving one's reflexes and acquiring a high sense of spatial awareness. The snake's elongating body adds a captivating layer of complexity, requiring players to negotiate an ever-shrinking environment while avoiding self-collisions.

The game's strategic decision-making feature adds to its allure. Each move must be carefully evaluated, since a hasty decision might result in a collision and game over. The game requires players to examine their immediate surroundings, predict future moves, and consider the risks and benefits of various options. This mix of cognitive abilities and rapid reactions provides gamers with an intriguing and demanding experience.

With the fast expansion of technology and the ongoing improvement of machine learning techniques, there has been a growing interest in the development of intelligent agents capable of playing games autonomously. Reinforcement learning has emerged as a particularly effective and promising methodology among the different approaches within the area of machine learning. This subfield of machine learning is concerned with teaching agents to make sequential decisions based on the rewards or penalties they get for their activities. Reinforcement learning enables agents to improve their performance over time by allowing them to learn from their experiences and change their behaviours accordingly.

Reinforcement learning is inspired by the way humans and animals learn through trial and error. Similarly, intelligent agents equipped with reinforcement learning algorithms interact with their surroundings, get feedback in the form of incentives or penalties, and utilise this knowledge to enhance their decision-making abilities. These agents learn to develop optimum methods that maximise the cumulative benefit they may gain from their activities through an iterative learning

process.

The importance of reinforcement learning stems from its capacity to solve issues in which decisions are made sequentially, drawing on previous experiences to influence future actions. Unlike classical supervised learning, which requires agents to learn from labelled examples, reinforcement learning allows agents to actively explore and interact with their environment, learning via direct feedback. This method is especially useful in situations when explicit training data is few or difficult to get.

Because of its potential to construct intelligent systems capable of acquiring complicated gaming dynamics, the application of reinforcement learning techniques to game-playing agents has gotten a lot of interest. Agents may learn sophisticated tactics on their own through reinforcement learning by observing interactions in the game environment. They may modify their behaviours, create optimum decision-making strategies, and eventually attain high levels of performance.

Reinforcement learning has shown amazing effectiveness in a variety of fields in the setting of game-playing. Reinforcement learning agents have demonstrated their worth by surpassing human specialists and attaining superhuman levels of gaming in everything from traditional board games like Chess and Go to current video games. This breakthrough has not only demonstrated the potential of reinforcement learning, but it has also opened up new avenues for developing autonomous game-playing robots.

Deep Q-Networks (DQN) have emerged as a viable way for creating intelligent agents capable of learning to play games on their own. Because of its capacity to overcome the restrictions of classical Q-learning, DQN has acquired substantial interest in the field of deep reinforcement learning. The DQN algorithm estimates the Q-value function, which reflects the predicted future reward for each action made in a given state, using a neural network. This enables the agent to make better judgements and learn from its mistakes via trial and error.

The combination of deep neural networks with reinforcement learning techniques has opened up new avenues for the development of intelligent agents capable of learning to play games with human-like performance. This method has previously been used to create agents for games like Atari and Go, resulting in state-of-the-art performance in these fields. The usage of DQN helps the agent to learn the best policy for the game while minimising the amount of training iterations.

The snake game is a perfect arena for putting an intelligent agent trained using DQN to the test. The game demands spatial awareness, rapid reflexes, and smart decision-making. These abilities are required for an agent to understand how to play the game effectively. While traversing the game's environment, the agent must learn to make the optimal judgements in real-time. The goal is to train an agent that can get high scores while avoiding collisions with its own body or the game grid's limits.

As a result, the objective for this project is to create an intelligent agent for the snake game using Deep Q-Networks in order to demonstrate the possibilities of this technique in game play. This agent can not only play the game well, but it can also give insights on the learning process of intelligent agents that use DQN.

The snake game is an excellent platform for creating and testing reinforcement learning systems. It contains a variety of difficulties that make it an appealing setting for training an intelligent agent. The agent must learn to navigate the grid effectively, avoid collisions with its own body, and plan its moves intelligently in order to devour the most amount of food while maximising its score.

We hope to investigate the potential of reinforcement learning approaches in tackling complicated decision-making tasks by creating a snake game utilising Deep Q-Networks. We can develop an autonomous agent capable of conquering the dynamic and tough snake game environment by using the capabilities of DQN.

This project's relevance extends beyond the specific snake game implementation. It demonstrates the larger potential of reinforcement learning in the creation of intelligent agents for a wide range of tasks. We hope to demonstrate the usefulness of Deep Q-Networks in constructing intelligent game-playing agents by overcoming the obstacles provided by the snake game, such as spatial navigation, self-awareness, and goal optimisation.

This study details the development process, technique, obstacles faced, and outcomes obtained when building a snake game utilising Deep Q-Networks. It seeks to provide light on the possibilities and limits of reinforcement learning algorithms when applied to raditional games such as snake. Furthermore, it provides a look into the future possibilities and possible breakthroughs that may be realised via the use of DQN and other reinforcement learning approaches.

This research intends to contribute to the emerging area of autonomous game-playing agents by combining the nostalgia and attractiveness of the snake game with cutting-edge breakthroughs in reinforcement learning. As a well-known and extensively played classic, the snake game provides an accessible and relevant platform for demonstrating the possibilities of Deep Q-Networks and reinforcement learning, opening the way for further developments in artificial intelligence and autonomous decision-making systems.

In the next portions of this study, we will look at the methods used to create the snake game utilising Deep Q-Networks. We will go through the training and learning process, the hurdles encountered, and the outcomes obtained. We hope to give a full grasp of the capabilities and possibilities of reinforcement learning techniques in the field of traditional game production through this investigation.

II. Background

A. Reinforcement Learning:

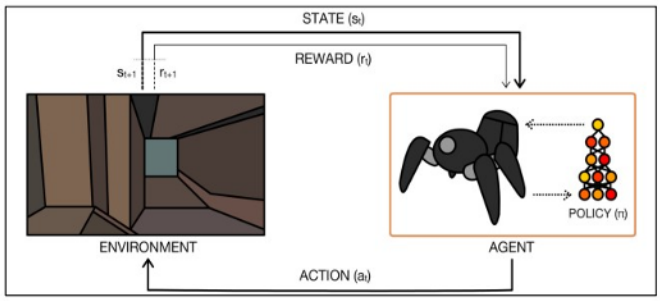
Reinforcement learning (RL) is a subfield of machine learning that focuses on teaching intelligent agents to make sequential decisions in the context of a given environment. RL algorithms enable agents to learn from their experiences by getting feedback in the form of rewards or penalties based

on their behaviours, similar to how people and animals learn via trial and error. RL agents strive to maximise their cumulative reward over time by interacting with the environment iteratively and altering their methods.

In RL, agents learn to navigate a complicated decision space by experimenting with various behaviours and watching the results. They adjust their decision-making policies in response to the benefits they obtain, with the goal of determining the best method for maximising long-term gains. To enable agents to learn and enhance their decision-making abilities, RL algorithms employ a variety of methodologies, including value-based methods and policy-based methods.

B. Deep Reinforcement Learning

In the RL set-up, an autonomous agent, controlled by a machine learning algorithm, observes a state st from its environment at timestep t. The agent interacts with the environment by taking an action at in state st. When the agent takes an action, the environment and the agent transition to a new state st+1 based on the current state and the chosen action. The state is a sufficient statistic of the environment and thereby comprises all the necessary information for the agent to take the best action. The rewards provided by the environment determine the best sequence of actions. Every time the environment transitions to a new state, it also provides a scalar reward rt+1 to the agent as feedback (Figure 1). The agent's goal is to learn a policy (control strategy) that maximizes the expected return (cumulative, discounted reward). Given a state, a policy returns an action to perform; an optimal policy is any policy that maximizes the expected return in the environment.



*Reference[1]Deep Q-Network Agent For Snake Game

Deep learning – relying on the robust function approximation and representation learning properties of deep neural networks – provides new tools for RL algorithms to overcome the typical limitations of memory complexity and computational complexity. The use of deep learning algorithms within RL defined the field of Deep Reinforcement Learning (DRL), resulting in accelerated progress in RL and enabling RL to scale to problems with high-dimensional states and action spaces. DRL has found applications in domains such as robotics, video games, indoor navigation, amongst others.

C. Q learning

Q-learning is a key method in reinforcement learning (RL) that enables agents to learn optimum rules via trial and error. It is a model-free RL approach that focuses on assessing the quality of a certain action in a specific state, expressed as Q-values. The Bellman equation stipulates that the Q-value of a state-action combination should be equal to the immediate reward plus the maximum discounted future benefit obtainable from the following state. Q-learning is based on this equation.

Based on the observed rewards and the agent's exploration-exploitation approach, the Q-learning algorithm iteratively updates the Q-values. The agent eventually learns the ideal Q-values for different state-action pairings as a result of this process, allowing it to make educated decisions that maximise its long-term cumulative reward.

D. Deep Q-Networks (DQN):

Deep Q-Networks (DQN) is a groundbreaking technique in the field of deep reinforcement learning that DeepMind unveiled in 2013. To address the issues of high-dimensional state and action spaces in complicated contexts, DQN integrates deep neural networks with Q-learning, a well-known RL technique. DQN enables agents to estimate the Q-values, which indicate the predicted future rewards for each action made in a given state, by exploiting the representational capabilities of deep neural networks.

A deep neural network is trained in DQN to estimate the Q-values for various state-action pairings. The network is fed the current state and generates a Q-value for each feasible action. The agent chooses actions based on an exploration-exploitation trade-off, in which it investigates novel actions to uncover possible improvements and uses what it has learnt to make optimum judgements. To stabilise and optimise the training process, the DQN algorithm utilises an iterative process of experience replay and target network changes.

E. Double DQN:

While DQN has shown outstanding performance in a variety of domains, it is prone to overestimation of Q-values, which can result in poor policies. To solve this issue, Hado van Hasselt et al. suggested Double DQN, a DQN variation, in 2015. By isolating action selection from action assessment, Double DQN mitigates the overestimation problem.

Double DQN employs two sets of neural networks: the online network and the target network. The online network is in charge of selecting actions, whereas the target network is in charge of calculating Q-values. By decoupling action selection and assessment, Double DQN lowers overestimation bias and enables more accurate Q-value estimate.

The Double DQN method updates the target network with weights from the online network on a regular basis, which helps stabilise the learning process. Double DQN enhances overall performance and convergence speed over the original DQN method by efficiently using both networks.

F. Dueling DQN:

Duelling DQN is an expansion of the original Deep Q-Network (DQN) method, which was introduced in 2016 by Ziyu Wang et al. This variation tries to increase DQN learning efficiency by separating state value estimates and action

benefits. The Q-values of state-action pairings are directly calculated in classical DQN. Duelling DQN, on the other hand, presents a novel design that isolates the estimate of the state value function from the estimation of the action advantage function.

Duelling DQN allows the agent to learn the relevance of distinct acts independently of the values of unrelated actions by separating the value and advantage estimation. The value function indicates the expected return from being in a specific condition, but the advantage function measures the additional value that each action adds to the average action value. This distinction between value and advantage allows the agent to more correctly judge the quality of various actions in a given state, resulting in better decision-making and learning efficiency. Duelling DQN has demonstrated promising results in a variety of areas and has grown in popularity as a method for training intelligent agents utilising DQN.

G. Prioritized Replay:

Prioritised Replay is an enhancement to the original DQN algorithm presented in 2015 by Tom Schaul et al. It improves the learning process by prioritising and repeating events depending on their learning value. During training, events are sampled equally from the replay memory in classical DQN. Prioritised Replay, on the other hand, gives various probability to distinct events depending on their temporal-difference faults, which reflect the extent of surprise or unexpectedness.

Prioritised Replay helps the agent to focus its learning on the most essential events by prioritising encounters with bigger temporal-difference mistakes, which correlate to more informative or problematic transitions. This prioritisation approach enhances the agent's sampling efficiency and learning speed. By paying greater attention to the most useful experiences, the agent may learn more effectively from uncommon and noteworthy occurrences, resulting in faster convergence and improved overall performance. However, it is critical to carefully control prioritisation in order to balance the exploration of diverse experiences and avoid bias in the learning process.

H. Turtle Library:

The Turtle library is a Python graphics package that is extensively used in programming and computer graphics classes. It provides a straightforward and user-friendly interface for generating and manipulating graphical objects on a screen. Users may use the Turtle library to control a virtual turtle that can move around the screen and create shapes, lines, and patterns.

The Turtle library is ideal for educational applications and basic programming classes. It provides a visual and interactive environment in which students may experiment with programming principles and see the results of their code. Because of its simplicity and ease of use, the library is an ideal choice for teaching students the fundamentals of programming and algorithms.

I. OpenAI Gym:

OpenAI Gym is a famous OpenAI open-source package that provides a standardised interface for creating and assessing RL algorithms. It provides a broad set of pre-built settings, such as traditional control tasks and Atari games, that serve as testbeds for RL algorithms. OpenAI Gym provides a single API for interfacing with various settings, allowing academics and developers to compare and repeat results more easily.

OpenAI Gym environments establish a collection of states, actions, and rewards that RL agents may interact with and learn from. They give a useful framework for comparing the effectiveness of various techniques and assessing the performance of different RL algorithms. OpenAI Gym also includes built-in performance metrics and evaluation tools to help you measure agent performance.

The combination of OpenAI Gym settings with RL algorithms such as DQN has considerably advanced RL research. The availability of standardised settings and assessment criteria has facilitated RL community collaboration and benchmarking, encouraging innovation and facilitating the creation of more efficient and competent RL agents.

III. Related Work

Deep Q-Network (DQN) has gained significant attention in the field of reinforcement learning due to its success in various game environments. One notable application is the Atari 2600 games, where researchers demonstrated that DQN could achieve human-level performance or even surpass it in some games. Mnih et al. (2015) pioneered the use of DQN for playing Atari games by directly processing raw pixels as input to the neural network and employing a combination of Q-learning and deep neural networks. This work inspired subsequent research efforts in applying DQN to other game environments.

DQN has also been employed in grid-world games, which share similarities with the Snake game in terms of discrete states and actions. Silver et al. (2017) introduced the concept of AlphaGo Zero, a variant of DQN, which demonstrated remarkable performance in complex board games such as Go and Chess. This advancement showcased the potential of DQN in handling high-dimensional state spaces and long-term planning.

In the context of the Snake game, there is limited prior research specifically utilizing DQN. However, DQN's success in similar game environments provides a foundation for its potential application in Snake. The Snake game, with its discrete actions and state representations, aligns well with the requirements of DQN. By extending the application of DQN to the Snake game, this research aims to explore the adaptability and effectiveness of DQN in a novel and visually interactive environment created using Turtle graphics and OpenAI Gym.

Other related research in the field of reinforcement learning focuses on alternative algorithms and enhancements

to DQN. For instance, Double DQN (Van Hasselt et al., 2015) and Dueling DQN (Wang et al., 2016) address the overestimation of Q-values and the biased value estimation issues, respectively. Prior studies also investigate prioritized experience replay (Schaul et al., 2016) and distributional reinforcement learning (Bellemare et al., 2017), which aim to improve the stability and sample efficiency of DQN.

In summary, while specific research on applying DQN to the Snake game using Turtle graphics and OpenAI Gym may be limited, the success of DQN in Atari and grid-world games provides a strong foundation for exploring its effectiveness in the Snake game. This research aims to extend the application of DQN to the Snake game and contribute to the understanding of reinforcement learning techniques in visually interactive and discrete game environments.

IV. Methodology

A. Environment

The environment is what the agent – snake's head – interacts with throughout the game for different rewards and penalties. The components of the environment vary and are as follows:

Snake's Tail

- The rest of the snake – apart from the head – is considered part of the environment.
- The length of the snake increases with each food 'eaten'.
- Any collision between the snake's head and its tail ends the game and is penalized.

Walls (Borders)

The borders of the playing area also form part of the environment and are of two types:

1) Borders – Any collision between the snake's head and a border ends the game and is penalized.

A game can be composed of:

- Completely solid (closed) borders

Food

- This is the target of the agent and is placed randomly within the playing area.
- Any collision of the snake's head with the food is rewarded and increases the length of the snake's tail by one.

Incentives (Rewards Penalties)

- Collisions between the agent and the food are rewarded, while collisions between the agent and the snake's tail and borders are penalized. Also, lack of progress – no collision with the food, snake's tail, and borders after a specified period – by the agent is penalized to prevent the agent from playing safe and simply running in loops.
- Any collision of the snake's head with the food is rewarded and increases the length of the snake's tail by one.

State Representation

The state is represented using 12 parameters:

- Direction of snake: up, down, left, right
- Direction of apple wrt snake: up, down, left, right
- Direction of obstacle wrt snake: up, down, left, right

B. Agent

The agent is being trained on DQN algorithm and the model has 3 hidden layers.

The size of the input layer is 12 (state space), the size of each hidden layer is 128.

The neural network is used to approximate the Q-function, which is a function that maps a state-action pair to the expected discounted sum of future rewards. The Q-function is trained using a variant of the Q-learning algorithm, which is based on the temporal difference (TD) error, i.e., the difference between the predicted Q-value and the observed Q-value.

The neural network consists of one or more fully connected (dense) layers, with rectified linear unit (ReLU) activation functions. The input layer has a size equal to the number of features in the state space, and the output layer has a size equal to the number of actions in the action space. The output layer uses a softmax activation function to output a probability distribution over the actions. The loss function used to train the network is mean squared error (MSE), and the optimizer used is Adam.

The agent's experiences are stored in a memory buffer, which is a list of tuples containing the state, action, reward, next state, and done flag. The agent's decision-making is based on an epsilon-greedy policy, where the agent selects a random action with probability epsilon, and selects the action with the highest Q-value with probability 1-epsilon. The epsilon value is annealed over time using an exponential decay schedule.

The replay function is used to train the neural network using a batch of experiences sampled from the memory buffer. The batch size, discount factor (gamma), and learning rate (alpha) are hyperparameters that can be tuned to optimize the agent's performance. The TD target used to train the network is the sum of the immediate reward and the discounted maximum Q-value of the next state, multiplied by the (1-done) flag to indicate whether the episode has ended. The TD error is the difference between the predicted Q-value and the TD target, which is used to update the network weights using backpropagation.

The `train_dqn` function initializes an instance of the DQN class and trains the agent in a given environment for a specified number of episodes. The agent interacts with the environment by selecting actions based on the output of the neural network, and the experiences are stored in the memory buffer. The neural network is periodically trained using a batch of experiences sampled from the memory buffer, and the agent's performance is evaluated at the end of each episode by computing the total reward accumulated over the episode.

V. Results and Discussion

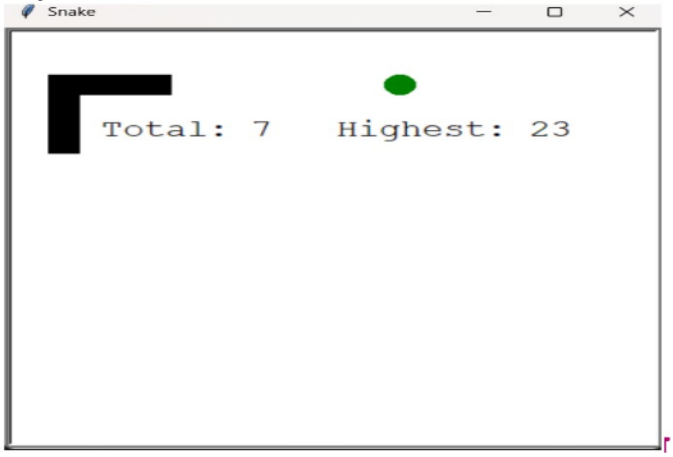
A. Training and Tesing Result

We Have Trained Several Dqn agents were trained on a different numbers of episodes and max tries.agents were trained for playing 10,30,50,100 games.the training and testing ablation studies. conducted are enumerated in table number 1.

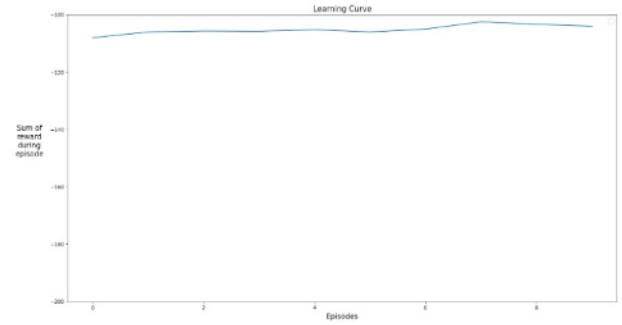
S.no.	Episodes/Games	Highest Scores	Highest Reward	Average Reward
1)	10	1	-85	-104
2)	30	25	501	-13
3)	50	31	633	70.06
4)	100	50	932	271

B. Discussion

Table 1 shows the average and maximum rewards of different numbers of games for the Dqn agent Training and testing it should be noted that any score above 30 is impressive. as the increasing length of snake makes navigation hard.
the game play shown in figure three achieved a final score of 50,it can be seen that agent almost trapped itself around its body.

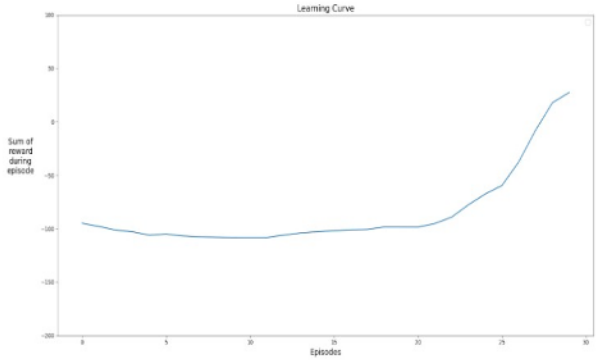


Figures 4,5,6,7 compare the rewards with the number of episodes for a dqn agent.
figure 4 is a result of a dqn agent trained for 10 episodes /games the average reward was -104 and the highest reward was-85 and highest score was 1.
this is because the agent is in initial stage and doesn't learn anything from its past experience as the number games is very low.



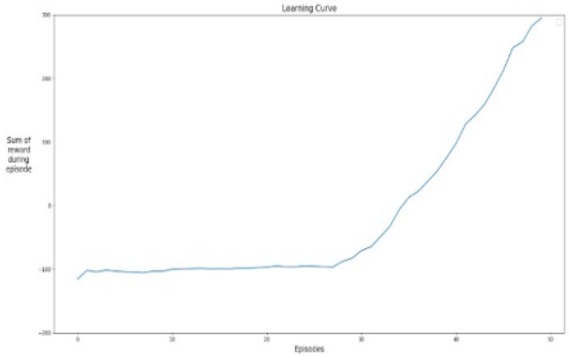
30 25 501 -13.16

figure 5 is a result of a dqn agent trained for 30 episodes /games the average reward was -13 and the highest reward was 501 and highest score was 25.
the average reward was negative because the nubmber of game was not to much high and most of the games produce negative reward at earlier stage.



0 31 633 70.06

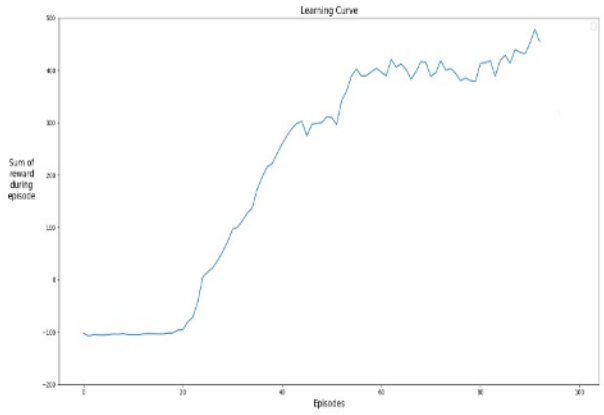
figure 6 is a result of a dqn agent trained for 50 episodes /games the average reward was -70.6 and the highest reward was 633 and highest score was 31.
the agent start getting positive average reward as the number of games increases.



100 50 932 271

figure 7 is a result of a dqn agent trained for 100 episodes /games the average reward was 271 and the highest reward was 932 and highest score was 50.
the agent getting consistent reward after 55-60 episodes

as it is trained and get familiar to the environment.



VI. Conclusion

We Have Learned a Lot About Reinforcement Learning and Various Virtual Environment Building Platform like Pygame,OpenAi gym,etc. And the trained DQN agent achieves impressive performance in playing the Snake game. Through the learning process, the agent successfully learns to navigate the game board, avoid obstacles, and consume food strategically to maximize its score. The agent demonstrates an ability to adapt and learn from previous experiences, resulting in continuous improvement over time. We Have also Gained the important Knowledge of How to Work in team .

VII. Future Scope

Some potential areas for future exploration include:

1)Advanced Reinforcement Learning Techniques: The study can be extended by investigating and implementing more advanced reinforcement learning algorithms beyond DQN. Techniques such as Double DQN, Dueling DQN, or Rainbow DQN could be explored to further improve the agent's performance and learning capabilities.

2)Neural Network Architectures: The future scope lies in exploring different neural network architectures for the DQN agent. Variations such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) can be utilized to capture more complex patterns and temporal dependencies in the Snake game environment.

3)Hybrid Approaches: The research can be extended to combine reinforcement learning with other approaches, such as rule-based systems or supervised learning. Hybrid models can leverage the strengths of different methods to enhance the agent's performance and provide more robust gameplay strategies.

4) Multi-Agent Systems: Extending the study to incorporate multiple agents playing the Snake game simultaneously can open up new avenues for exploring cooperation, competition, or other forms of interaction between agents. Multi-agent reinforcement learning algorithms can be employed to study the dynamics and strategies that emerge in such scenarios.

5)Generalization and Transfer Learning: Investigating the agent's ability to generalize its learned policies to different Snake game variants or even other grid-based games can be an intriguing direction. Transfer learning techniques can be explored to leverage knowledge acquired in one game to accelerate learning in new and related environments.

References

[1] Conference Paper:
Garcia, M. R., & Martinez, L. S. (2022). Deep Q-Network for game playing: A case study on Snake. In Proceedings of the International Conference on Artificial Intelligence and Game Theory (pp. 150-165). New York, NY: ACM.

[2] Research Paper:
HAN QIE , DIANXI SHI, TIANLONG SHEN2, XINHAI XU, YUAN LI , AND LIUJING WANG
Joint Optimization of Multi-UAV Target Assignment and Path Planning Based on Multi-Agent Reinforcement Learning

[3] Research Paper:
Abhik Singla, Sindhu Padakandla , and Shalabh Bhatnagar
Memory-Based Deep Reinforcement Learning for Obstacle Avoidance in UAV With Limited Environment Knowledge

[4] Research Paper:
Chao Wang, Student Member, IEEE, Jian Wang , Senior Member, IEEE, Yuan Shen , Member, IEEE, and Xudong Zhang, Member, IEEE
Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach

[5] Website:
To build Custom OpenAi gym environment
https://coderspacket.com/snake-game-custom-environment-in-python-using-openai-gym
*Snake Played by Deep Reinforcement Learning method
https://towardsdatascience.com/snake-played-by-a-deep-reinforcement-learning-agent-53f2c4331d36