21AIE401 - Deep Reinforcement Learning

Snake Game using Deep Reinforcement Learning

Batch: AIE - A **Team Info** - Raja Pavan Karthik - AM.EN.U4AIE20060

Group - 9 Sista Sai Subramanya Mrinaal - AM.EN.U4AIE20067

Introduction: This project aims to develop an AI model capable of learning how to play the popular game Snake from scratch using Deep Reinforcement Learning. The model starts with no prior knowledge and must learn on its own to maximize the score or reward by formulating an effective strategy. The agent receives state information about the game environment and learns to make decisions without explicit instructions. Through the Deep Q-Learning algorithm, the agent identifies optimal strategies to maximize its in-game score. The AI learns to navigate the snake to find and consume food while avoiding collisions with boundaries and its own body. As the agent successfully finds food, the snake grows. However, if the snake crashes into the boundaries or itself, the game ends. This project will teach you about the most important parts of DRL, including Q-learning, DQN architectures, experience replay, and reward engineering.

Related works:

[1] This paper explores using Deep Q-Learning to teach AI how to play the Snake game, addressing the challenge of training AI for classic games. It discusses reinforcement learning concepts, emphasizes neural network hyperparameter tuning, and presents numerical simulations for training and testing. The goal is to achieve strong performance in playing Snake.

[2] This research explores the challenges of analyzing Reinforcement Learning applications, particularly when combined with Deep Reinforcement Learning and neural networks. It focuses on training a snake game agent using Python, Keras, sci-kit image, and TensorFlow, with SARSA controlling snake movements and Deep Q-learning in the learning module. The study also highlights the significance of the SARSA algorithm's learning rate in influencing the agent's performance across different iterations.

[3] This project aims to build an AI bot that learns to play the game Snake from scratch using Deep Q Learning, a form of Unsupervised Machine Learning. The bot receives rewards based on its actions in the game and has no prior knowledge of the game rules. The goal is for the bot to figure out how to play and maximize its score. Various strategies, such as memory storage, random actions, and Convolutional Neural Networks, will be explored to improve its learning process.

[4] In this project, we use Reinforcement Learning, specifically a Deep Q-Network (DQN), to play Snake. However, DQN can suffer from overfitting, causing lower scores after extensive training. To counter this, we improved rewards, provided the snake more environmental data, and discussed optimizing DQN compared to other algorithms.

[5] This paper presents a streamlined Deep Reinforcement Learning (DRL) approach that works well with compressed image data, without requiring extra environment information and with reduced memory and

time demands. It uses a lightweight Convolutional Neural Network (CNN) and a simple reward system, achieving performance similar to other DRL methods in playing the Snake game.

[6] This paper improves Deep Reinforcement Learning (DRL) for playing the Snake Game, a challenging scenario with delayed rewards. It uses a Convolutional Neural Network (CNN) with Q-learning, a smart reward system, adapts to changing targets with a training gap strategy, and enhances training with a dual experience replay method. Results show the agent surpasses human-level performance and outperforms the baseline model in Snake.

Methodology:

The game itself is constructed using Pygame, incorporating a game loop that repeatedly executes the game within a continuous cycle. The overall system comprises an Agent that employs Deep Q-Learning to make decisions within a Snake game. The Agent actively interacts with the game through an Environment, which provides the Agent with state information and rewards. The rewards system assigns a value of +10 for successfully consuming food, -10 for colliding with obstacles, and 0 for any other scenario. The available actions for the Agent encompass moving straight, to the left, or to the right. The Deep Q-Learning process commences by initializing Q-values, and subsequently makes choices for actions, updating the Q-values through the utilization of the Bellman Equation. The model is then trained in a continuous loop, aiming to enhance the overall performance of the Agent.

Creating the game involves setting up all the required steps and procedures to be. We define the rewards as whenever a snake eats a food we give the +10 and when the game is over or it gets crashes it gives -10 and for everything else it gives 0, Additionally the environment contains negative reward -5, when the snake eats it the score gets decreased by 5. We add some rewards with no increasing score i.e when the snake eats it there's no change in score or can be considered as null but we added these to increase the efficiency of the agent and hardness of the game. The actions in the game are straight, left, right of the snake. All of these are defined in 3 parameters which helps to control the snake properly. If we use the single parameter values the turning of the snake might be difficult. These are the actions with their values [1,0,0] for straight, [0,1,0] for right turn and [0,0,1] for left turn. In order to train the model we need to feed the agent about the game environment. Defining the states in the game is crucial so here there exists 11 - states. All these are boolean values.

Up: This state corresponds to the snake's upward movement on the game grid.
Right: In this state, the snake is traveling to the right.
Left: The snake is in motion to the left within this state.
Down: In this state, the snake is making a downward movement on the grid.
Danger Up: When the snake is in this state, it's on the brink of a collision or facing a potential danger if it continues moving upward.
Danger Right: In this state, the snake is at risk of colliding if it proceeds to move to the right.

Danger Left: Here, the snake is in a situation where it might collide if it decides to move to the left.

Danger Down: The snake faces the possibility of a collision when it moves downward in this state.
Food Up: This state indicates that the snake has detected food and is making its way towards it by moving upward.
Food Down: In this state, the snake has detected food and is heading toward it while moving in a downward direction.
Food Left: When the snake is in this state, it's moving to the left in order to reach the detected food.
Food Right: The snake is moving to the right to reach the detected food while in this state.

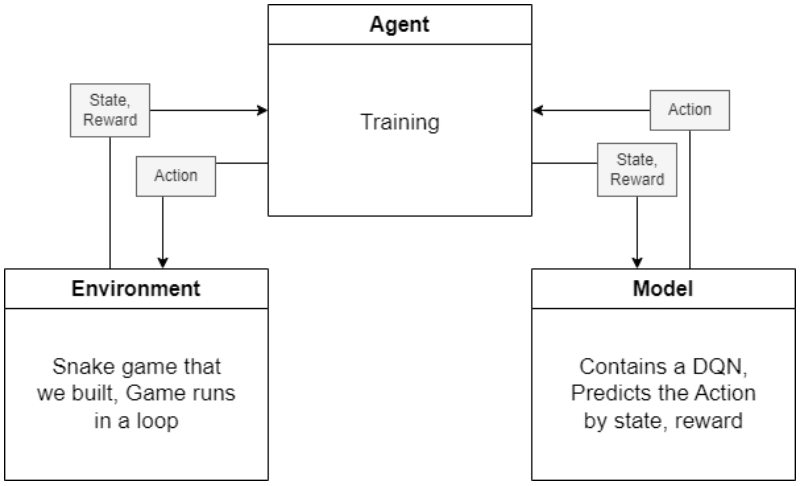

Figure 1: Function Module Diagram

We successfully initialized the classic Snake game using Python and the Pygame library. It starts a game window and sets up the game state, including the snake's initial position, direction, and the location of the food. In each game step, it collects user input to change the snake's direction, updates the snake's position, not only that but it also checks for collisions with the boundaries or itself, and manages scoring. The game env displays the snake, food, and the current score on the screen. Which runs in an infinite loop until the snake collides with an obstacle, at which point the final score is printed, and the game window is closed. This code forms the foundation of a basic Snake game.

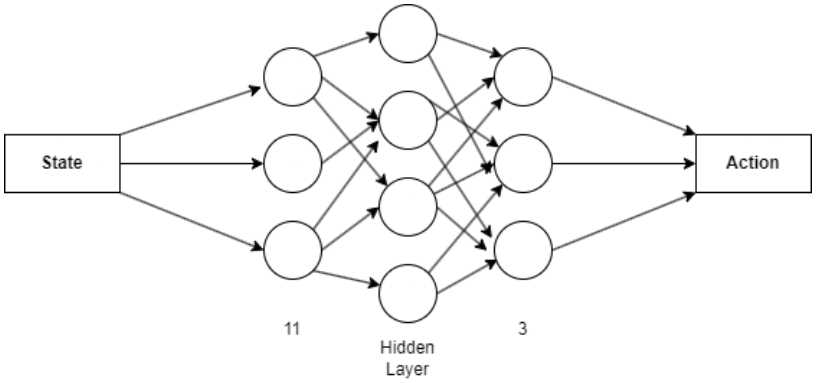

Figure 1: Function Module Diagram

Algorithm/Working:

- ❖ **Agent**
- ➢ Model (PyTorch)
    - ○ Linear Q Net (DQN)
    - ○ Model.predict(State)
        - ■ Action
- ➢ Game (Pygame)
    - ○ Play Step(Action)
        - ■ Reward, Game Over, Score
- ❖ **Training**
- ➢ State = get state(Game)
- ➢ Action = get move(State)
    - ○ model.predict()
- ➢ Reward, Game Over, Score = Game.play step(Action)
- ➢ New State = get state(Game)
- ➢ Store/Save
- ➢ Model.train()

The Game is designed in a loop such that in each and every loop we do a play step function that takes an action which gives the direction to the snake to move in the created environment. Agent returns the current reward and also if the game is finished or not with total score. Here the agent puts everything together (model, game), the agent should know about the game, model. Then we can train the model. The training initially begins with calculating the state. Based on the state we can calculate the action with the help of a model. After getting the value of action we can start playing the game which returns the reward, final state, score. When the action is completed we get three values which help in finding the new state. We should not forget to save all of these inorder to train the model well (new state, old state, final state, score). With these we can train the model called linear DQN - here it is a feed forward neural network with a few linear layers which need the (new state, old state, final state, score) to predict the next action better.

In the context of training a Deep Q-Learning model for the Snake game, Initialization of Q-Values, The model initialization step marks the commencement of the training process. At this stage, the Q-values, which represent the expected cumulative rewards for taking various actions in different states, are initialized. To start, the model must choose an action. Initially, there is no specific state, as it involves a trade-off between exploration and exploitation. The chosen action is often determined through the model's predictions. The selected action is performed within the game environment, and the consequences of this action become apparent. Following the execution of an action, the model measures the immediate reward obtained from that action. The reward can be positive, negative, or zero, and it serves as feedback to guide the learning process.

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma\, maxQ'(s',a') - Q(s,a)]$$

Bellman Equation

After measuring the reward, the Q-value for the corresponding state-action pair is updated. This update is achieved through training the model iteratively. The specific algorithm ensures that the Q-values approximate the true expected returns for each state-action pair. The entire process is performed iteratively, where the model repeatedly selects actions, measures rewards, and updates Q-values until it converges to an optimal policy. To train the model effectively, a loss function is employed to optimize or minimize the discrepancies between the predicted Q-values and the values derived from the Bellman equation. The Bellman equation plays a pivotal role in reinforcement learning, enabling the model to update its Q-values according to the principle of optimality.

Results:

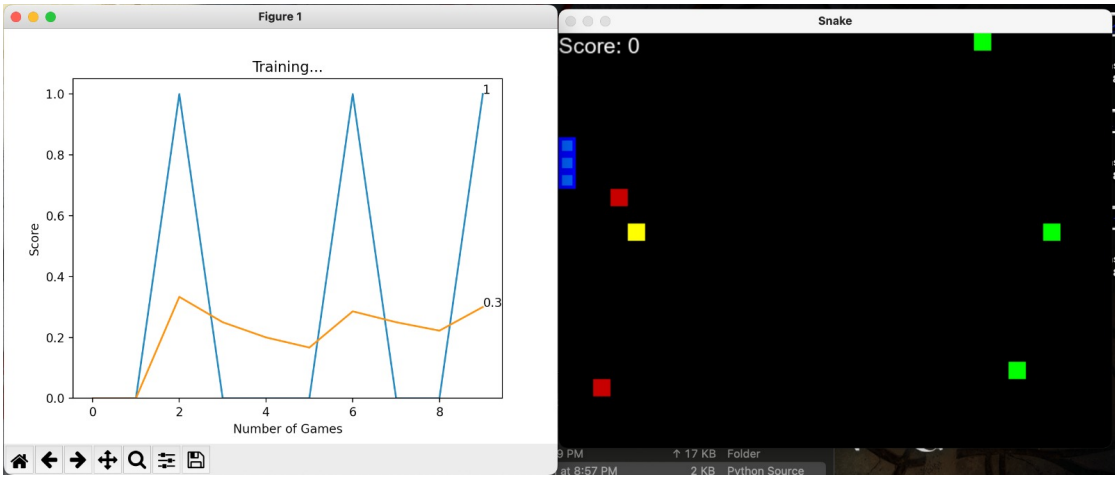

Figure 3: Initial stage of model training

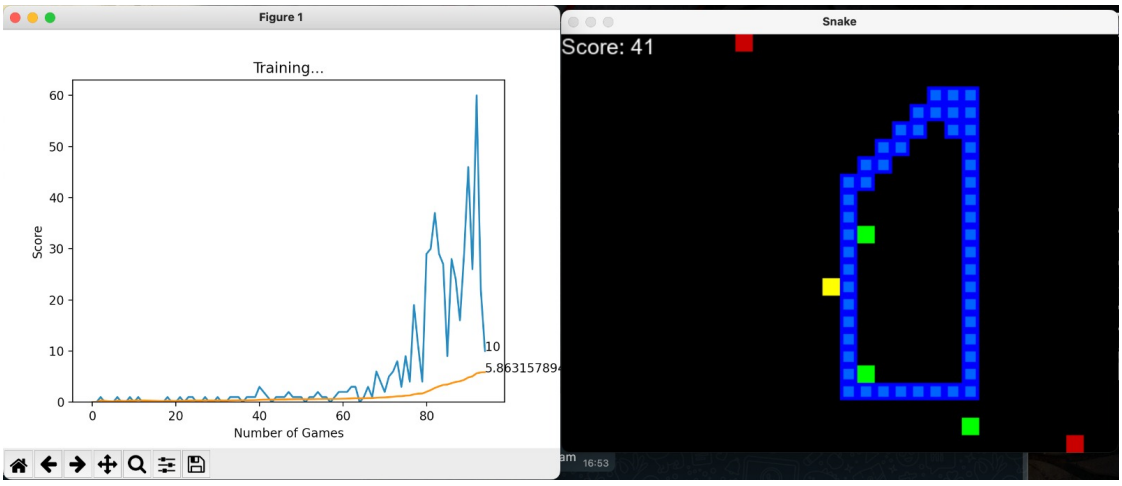

Figure 4: Model after trained for 12 mins

Observations: The code we have given different types of food (positive, negative, and yellow) adds complexity to the game. Players or AI agents must now make strategic decisions about which foods to eat and which to avoid. Each type of food is associated with a different reward or penalty. Positive food

provides a reward of 10 points, negative foods result in a penalty of -5 points, and yellow food gives a dummy reward of 0 points. This variety of rewards makes the game more interesting and challenging. The game display has been updated to visually represent the new types of food using distinct colors. Players can easily identify and differentiate between the different food items during gameplay. To maintain unpredictability, the new food items are randomly placed on the game board, with checks to ensure they do not overlap with the snake or other food items. These are particularly useful for training reinforcement learning agents. The game provides a wider range of rewards and penalties, allowing AI agents to learn more complex strategies. For human players, the new features add variety and unpredictability to the game, making it more engaging and challenging.

Future Scope: Expanding the Snake game to include multiple agents offers exciting opportunities for development. This multiplayer version can introduce new dynamics and challenges, such as multi-agent interaction, collaborative play, and team-based gameplay. In multi-agent interaction, designing rules for snake interactions, like collision avoidance and resource competition, will deepen the gaming experience. Collaborative play can test the coordination of snakes working together to achieve common objectives. Applying reinforcement learning to train AI-controlled snakes for competition or cooperation is a promising avenue. Team-based gameplay involves grouping snakes into teams with shared goals, fostering teamwork. Variability in game modes with different snake numbers and team configurations will enhance player experiences. Developing visualization and user interfaces for monitoring multiple snakes can improve gameplay. Expanding the game to include multiple agents can lead to dynamic and engaging gaming scenarios. It serves as an ideal platform for experimenting with multi-agent systems, reinforcement learning, and collaborative AI, making it valuable for both gaming and research.

References -

[1] A. Sebastianelli, M. Tipaldi, S. L. Ullo and L. Glielmo, "A Deep Q-Learning based approach applied to the Snake game," 2021 29th Mediterranean Conference on Control and Automation (MED), PUGLIA, Italy, 2021, pp. 348-353, doi: 10.1109/MED51440.2021.9480232.

[2] A. J. Almalki and P. Wocjan, "Exploration of Reinforcement Learning to Play Snake Game," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 377-381, doi: 10.1109/CSCI49370.2019.00073.

[3] Patil, Yashshree. “Snake Game Using Reinforcement Learning.” Academia.edu, January 13, 2021. https://www.academia.edu/44900555/Snake_Game_Using_Reinforcement_Learning.

[4] P. Yuhang, S. Yiqi, M. Qianli, G. Bowen, D. Junyi, and T. Zijun, “Playing the Snake Game with Reinforcement Learning”, *CEAS*, vol. 1, no. 1, Jul. 2023.

[5] M. R. R. Tushar and S. Siddique, "A Memory Efficient Deep Reinforcement Learning Approach For Snake Game Autonomous Agents," 2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT), Washington DC, DC, USA, 2022, pp. 1-6, doi: 10.1109/AICT55583.2022.10013603.

[6] Z. Wei, D. Wang, M. Zhang, A. -H. Tan, C. Miao and Y. Zhou, "Autonomous Agents in Snake Game via Deep Reinforcement Learning," 2018 IEEE International Conference on Agents (ICA), Singapore, 2018, pp. 20-25, doi: 10.1109/AGENTS.2018.8460004.