# CS 461 - Project Final Report

*Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey*
Emre Can Şen- 21902516, Ata Özlük - 21702335, Uygar Onat Erol - 21901908, Hande Eryılmaz - 21902678,
Mete Ertan - 21903215

***Abstract*— The objective of this project is to perform experiments using various Deep Reinforcement Learning methods to the "Snake" game. However, unlike the traditional game, our version incorporates a poison apple system. In this environment, our research compares the performances of Deep-Q-Networks, Double Deep-Q-Networks, and Dueling Deep-Q-Networks across each method. The outcomes of these methods are presented & analyzed.**

## I. Introduction & Significance

The project focuses on the experimental application of various Deep Reinforcement Learning methods to a modified version of the classic 'Snake' game, introducing a novel poison apple system. In contrast to the traditional gameplay, our adaptation introduces a stochastic element, compelling agents to not only pursue rewards but also strategically navigate potential pitfalls.

We re-implemented pivotal reinforcement learning models, including Linear Neural Network, Deep-Q Network (DQN), Dueling Deep-Q Network, and Double Deep-Q Network, within an established game framework. The detailed re-implementation steps of each of these methods are given in the methods section. The results obtained with Deep Q-Network, Double Deep Q-Network and Dueling Deep Q-Network were plotted and analyzed.

The addition of a poison apple system transforms the Snake game into an ideal testing ground for evaluating the adaptability of these models in a dynamic gaming scenario. Thus, our primary objective is to train the agent to achieve the highest score possible using these methodologies, shedding light on their individual strengths and weaknesses within the context of our modified game. The subsequent sections detail the meticulous re-implementation steps and present the results, offering valuable insights into the efficiency of Deep RL algorithms amidst the challenges posed by the poison apple system in our dynamic gaming environment.

## II. Literature Review

For the literature review, four primary academic papers were chosen, and their contents were analyzed. The first paper is "Playing Atari with Deep Reinforcement Learning" [1]. In this paper, the authors demonstrate the successful application of deep reinforcement learning to playing Atari 2600 games. The DQN algorithm presented in the paper combines deep neural networks with Q-learning to learn effective strategies directly from pixel inputs. The principles outlined in this paper provide a foundational understanding of how DRL can be applied to game environments, serving as a basis for subsequent advancements.

The second paper is "Deep Reinforcement Learning with Double Q-learning" [2]. Building upon the original DQN framework, this paper introduces Double DQN, a modification aimed at reducing overestimation bias in Q-learning. The authors address the issue of overestimating action values by decoupling the selection and evaluation of actions. Considering the potential benefits of Double DQN, its implementation in a Snake game may lead to more stable and accurate Q-value estimates, thereby improving the learning efficiency of the agent.

The third paper is "Dueling Network Architectures for Deep Reinforcement Learning" [3] Dueling DQN, proposed in this paper, introduces a novel architecture that separates the estimation of state values and action advantages. This modification enhances the learning process by allowing the agent to focus on states and actions that are most critical for decision-making. In the context of a Snake game, the Dueling DQN architecture may offer advantages in terms of better representing the value of different actions in various game states, potentially leading to more efficient learning.

The final paper is "Rainbow: Combining Improvements in Deep Reinforcement Learning" [4] Rainbow is a comprehensive integration of several improvements in deep reinforcement learning, including Double DQN, Dueling DQN, and additional enhancements such as prioritized experience replay. The Rainbow algorithm demonstrates superior performance across a range of Atari 2600 games. Applying the Rainbow algorithm to a Snake game project could potentially lead to a more robust and adaptable agent, leveraging the combined strengths of various DRL techniques. The methods of this paper were not re-implemented but extensive discussion regarding combining DRL to achieve more comprehensive methods provides useful insight for further improvements that can be made to the project.

The selected papers provide a solid foundation for implementing deep reinforcement learning techniques in our project. The principles from "Playing Atari with Deep Reinforcement Learning" inspire the use of DRL, while "Deep Reinforcement Learning with Double Q-learning" and "Dueling Network Architectures for Deep Reinforcement Learning" offer modifications that can enhance learning efficiency. Finally, "Rainbow: Combining Improvements in Deep Reinforcement Learning" presents a comprehensive approach that combines multiple advancements, potentially leading to a more powerful and effective Snake game-playing agent as further future improvements.

## III. METHODOLOGY

### A. Game Environment

As previously highlighted, our project employed an established game environment with specific adaptations to align with our objectives. The chosen environment was developed using the Pygame library, and we introduced modifications to incorporate a reward system. Rewards were assigned for various in-game events, including obtaining poisonous food, colliding with itself or the wall, and consuming normal food. The overarching aim was to optimize the snake's behavior to accumulate the maximum food-related rewards swiftly, thereby achieving the highest overall score. The resulting scores were dynamically displayed on the game window, providing a real-time representation of the agent's performance.

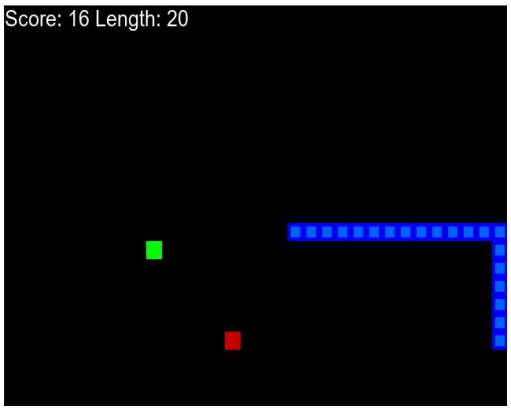


Fig. 1: Game Environment

Above, the window is labeled "Deep Q Network" and shows a blue snake with a length of 20 and a score of 16. The snake is currently moving horizontally across the black background, and there are two squares on the screen: one red and one green, which represents the food and the poisonous food, respectively.

The Dueling Deep Q Network and Double Deep Q Network runs on the same environment.

### B. Game Rules & Reward System

In this game, the snake's movement is confined to specific directions—up, down, left, or right—controlled by the agent, with no ability to move backward; directional changes are the only option. The agent maneuvers the snake strategically as it encounters randomly placed non-poisonous and poisonous food items on the screen. Consuming non-poisonous food elongates the snake and contributes +4 to the score. Whereas poisonous food, while not altering the snake's length, imposes a penalty of -1 on the agent. The ultimate objective for the player-controlled agent is to navigate these dynamics to maximize points, emphasizing both survival and strategic point accumulation. The game concludes if the snake collides with the game screen boundaries or itself, resulting in a termination penalty of -10.

### C. Poisonous Food

In our customized game environment, we introduced the concept of poisonous food alongside the pre-existing normal food items. Both types of food were implemented using a similar methodology. Notably, precautions were taken in the environment code to prevent collisions between the snake and normal food, ensuring a smooth and predictable gameplay experience. However, a potential issue arose with the possibility of collisions between poisonous and normal food. To address this, we implemented measures to ensure that these two types of food would not spawn at the same location, mitigating the risk of unintended interactions and maintaining the integrity of the game mechanics.

### D. Neural Network

#### 1) Linear Neural Network for Q Algorithms

We were supposed to implement a Convolutional Neural Network and the agent would use partial observability to observe its surroundings. However, this was not a feasible solution and the agent that worked with CNN took very long to make one move (around 0.3s). Since the agent makes many moves while training itself, we decided to use another approach. Our agent used a linear neural network for the final reports. Each move of the snake took around 1 ms with the linear neural network approach, which is 300 times faster than CNN approach.

#### 2) Linear Neural Network Model

The Linear Neural Network we used takes 11 input states. 4 of the states are the directions of the snake agent such as north, east. 4 of the states are the direction of the food and 3 of the states are the directions of collisions. We use 1 less state for the direction of the collision. If the agent snake makes an opposite direction move to its direction, it collapses all the time with itself unless the agent has a length of 1 and the agent starts with 3 length. Therefore, we do not need to put the snake agent's opposite direction to our output state and to the input of collapsing directions. Making collapse direction 3 states improves Q algorithms speed.

There are 3 output states in our Linear Neural Network, which are the directions in which the agent can move excluding its opposite direction. Excluding the opposite direction of the agent not only helps the algorithm and code run faster but also increases the total score the agent gets since we do not include a state output, which 100% of the time kills our agent.

### E. Deep Q-Network

Deep Q-learning is a type of reinforcement learning algorithm that combines Q-learning with deep neural networks. Deep Q-learning introduces deep neural networks to approximate the Q-function, which is a mapping from states and actions to Q-values. The neural network takes the state of the environment as input and outputs Q-values for each possible action. This allows the algorithm to handle high-dimensional input spaces, making it suitable for tasks like image-based decision-making in video games.

The key idea behind Deep Q-learning is to use the neural network to generalize across similar states and actions, enabling the agent to learn a more efficient and effective policy. The algorithm uses a technique called experience replay, where past experiences are stored in a replay buffer. During training, batches of experiences are sampled randomly from the buffer to train the neural network, reducing correlations in the data and improving stability.

The Target Q-Network used will be a Linear Neural Network provided in the previous section. Also the temporal loss is calculated then used for backpropagation.

### F. Double Deep Q-Network

Double Deep Q-learning is an extension of the original Deep Q-learning algorithm that aims to address a common issue known as overestimation bias [5]. In the standard Deep Q-learning algorithm, the Q-value is updated using the maximum Q-value of the next state. However, during the learning process, the Q-values themselves are also being updated, and this can result in an overestimation of their true values. The basic idea behind Double DQN is to decouple the selection of actions from the evaluation of those actions [5]. The algorithm modifies the Bellman equation as follows, rather than utilizing the same one as in the DQN algorithm:

$$Q(s, a; \theta) = r + \gamma Q(s', argmax_{a'} Q(s', a'; \theta); \theta')$$

Double Q-Learning re-implementation with Deep Neural Networks is called Double Deep Q-Network. The target neural network assesses this action to determine its Q-value after the main neural network θ selects the optimal next action a' from among all of the options. This method has been demonstrated to lower overestimations, leading to improved final policy [6].

In the Double DQN, the Q-network is used to select the best actions for the next states (next_actions). These actions are then used to index into the target Q-values obtained from the target Q-network (target_Q_values_selected). This separation of action selection and target evaluation is the key idea behind Double DQN and is intended to mitigate the overestimation bias that can occur in standard DQN. The primary difference is that Double DQN uses the Q-network for action selection and the target Q-network for evaluation of the selected actions during the computation of target Q-values.

### G. Dueling Deep Q-Network

This algorithm splits the Q-values into two different parts, the value function V(s) and the advantage function A(s, a).
The amount of reward that we will receive from state s is indicated by the value function V(s). Furthermore, the advantage function A(s, a) indicates the relative superiority of each action over the others. The Q-values can be obtained by combining the value V and the advantage A for each action:

$$Q(s, a) = V(s) + A(s, a)$$

By splitting the neural network's final layer into two parts—estimating the advantage function for each action a (A(s, a)) and the state value function for state s (V(s))—the Dueling DQN algorithm suggests that the neural network ultimately combines both parts into a single output that estimates the Q-values [6]. This modification is beneficial because, in many situations, understanding the state-value function alone may be enough rather than knowing the precise value of each action [6]. However in the standard DQN, both the state's value (V(s)) and the advantage of taking an action (A(s, a)) are considered when figuring out the Q-value. However, if the state is bad and all actions lead to failure, estimating the impact of each action is pointless since the state's value (V(s)) has already been calculated.

This design speeds up training because it only calculates the value of a state without having to calculate Q(s, a) for every action in that state. By separating the estimation between two streams, we can find more dependable Q values for each action. We can measure the model's loss using the following mean squared error:

$$L(\theta) = 1/N \sum (Q_\theta(s_i, a_i) - Q'_\Theta(s_i, a_i))^2$$

also where,

$$Q'(\theta) = R(s_t, a_t) + \gamma \max_{a'_i} Q_\theta(s'_i, a'_i)$$

then take the gradient descent step to update the model parameters.

Both DQN and Double DQN use a Deep Q-Network structure, whereas Dueling DQN uses a specialized Dueling Deep Q-Network structure to re-implement the dueling architecture.

In Dueling Deep Q-Network structure, the architecture explicitly separates the value and

advantage streams, allowing for independent estimation of the state value and advantages for each action. There is also a specialized Linear Neural Network reimplemented for Dueling DQN algorithm. This separation is intended to enhance the stability of the learning process. Other than the Dueling Deep Q-Network structure, the remaining code re-implementation is similar to DQN re-implementation.

## IV. Results & Analysis

The implementation of a reward system aligned with the project objectives effectively guides the snake's behavior. By assigning rewards for different in-game events, such as obtaining poisonous food, colliding with itself or the wall, and consuming normal food, the training process becomes goal-oriented. The real-time display of scores provides a transparent representation of the agent's performance.

Below are the results for each agent, including their respective Score Value, (eaten) Apple Count and (eaten) Poisonous Apple Count and their respective Mean Values.

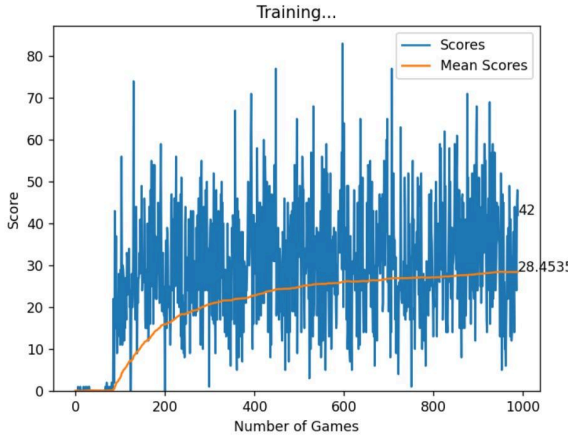


Fig. 2: Score per episode for DQN algorithm

Fig. 2 illustrates the scores per episode for the DQN algorithm.

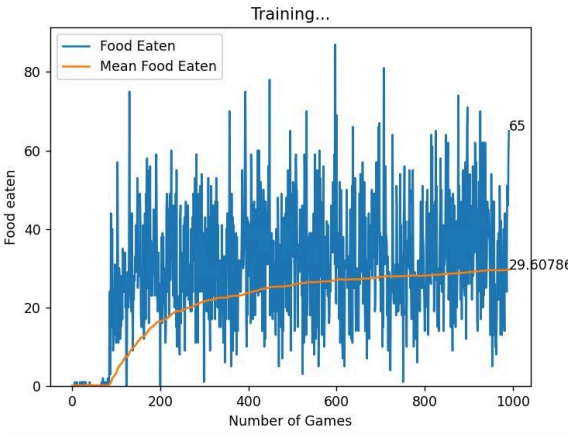


Fig. 3: Apples per episode for DQN algorithm

Fig. 3 delves into the quantity of apples consumed. Together with Fig. 4, these figures offer a comprehensive illustration of the reward dynamics within the specified method. They also serve as a valuable basis for a nuanced discussion on potential enhancements to the learning trajectory of the DQN agent. Insights derived from these figures prompt considerations on improving the agent's performance, whether through refining its approach to consuming poisonous apples or enhancing its avoidance strategy.

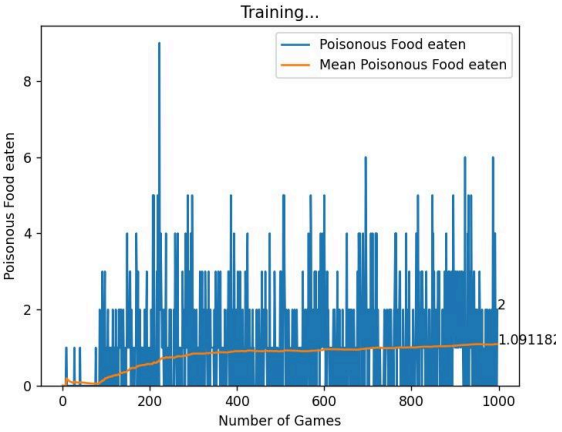


Fig. 4: Poisonous Apples per episode for DQN algorithm

Fig. 4 presents a detailed account of the poisonous apples consumed. An intriguing observation arises as the DQN agent consumes 9 poisonous apples within relatively short epochs, corresponding to a shorter snake length. This aspect raises concerns about the agent's efficiency in navigating the game dynamics. Notably, around the 200th epoch, the DQN agent ingested a total of 8 poisonous apples alongside 46 normal apples, resulting in a 14% ratio of poisonous apples eaten. While this ratio may not be a critical concern, given the competitive performance of DQN compared to other methods, it warrants further investigation and potential optimization to align with overall project objectives.

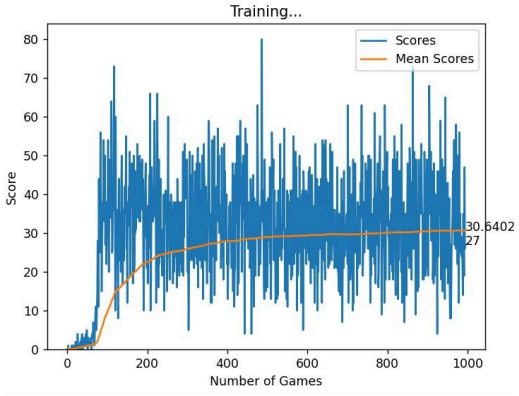

Fig. 5: Score per episode for Dueling DQN algorithm

In figure 5, the line labeled "Mean Scores," which indicates it is showing the average score over time. The line starts at zero and gradually increases, showing that our agent gets better scores while the number of games increases. This suggests that the average score is improving over time, which implies both consistent learning and optimization. The Dueling DQN algorithm as Figure 5 suggests gives better scores with the same number of games when compared to the DQN algorithm.

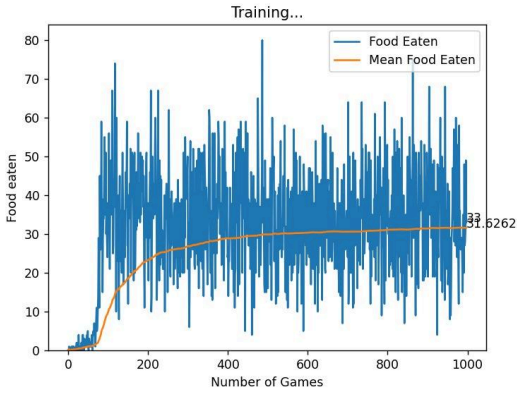


Fig. 6: Apples per episode for Dueling DQN algorithm

The "Mean Food Eaten" throughout several games is represented by the orange line in Figure 6. Additionally, the line shows that the quantity of food consumed increases in tandem with the number of games played. This significant tendency suggests that the agent becomes more proficient and successful in the given environment as a result of its learning skills becoming stronger with time.A learning curve where the agent gets better at navigating the game and getting food is suggested by the positive link between the number of games played and the amount of food consumed. The steady increase in the average amount of food consumed suggests that the agent is going through a learning process in which it is improving its tactics, developing its ability to make decisions, and adjusting to the demands of the gaming environment.

As a result, the agent shows a better capacity to catch and eat more food as the number of games rises, indicating an improvement in its ability to learn and make decisions throughout the duration of the observed games.

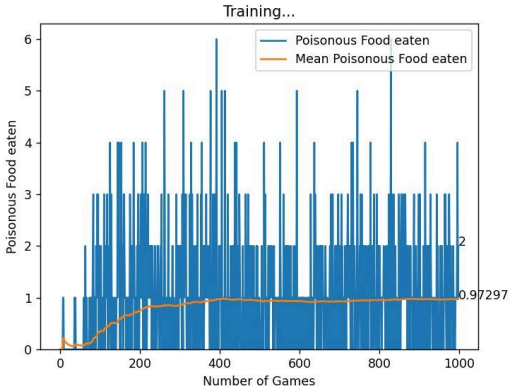


Fig. 7: Poisonous Apples per episode for Dueling DQN algorithm

Figure 7 shows the average amount of poisonous food consumed by the agent as an orange line. It indicates that the agent eats more poisonous food as the games progress. This increase primarily happens because the agent gets longer in the game. The agent's main goal is to avoid hitting walls or on itself, which gives a high penalty of -10. To avoid this, the agent chooses a smaller penalty by eating poisonous apples (-1) in order to not hit any obstacles and survive. Survival is the top priority, so the agent accepts a lower score to stay alive. This choice is based on the belief that the benefits of surviving longer outweigh the negative impact of eating poisonous food, allowing the snake to keep playing.

We can see that the introduction of poisonous food introduces complexity to the game environment. The precautions taken to prevent collisions between normal and poisonous food demonstrate a thoughtful approach to maintaining game integrity. This addition expands the decision space for the agent, requiring it to learn and adapt to both positive and negative rewards.

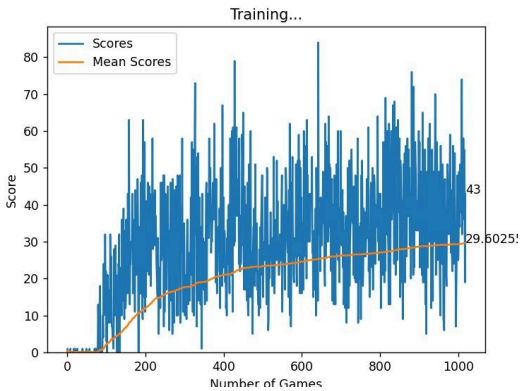


Fig. 8: Score per episode for Double DQN algorithm

The Double DQN algorithm's average agent score's temporal evolution is shown by the orange line in Figure 8. Starting with a starting point score

of zero, the line gradually rises, showing a noticeable increasing trend as the quantity of games played increases. The average score attained by the Double DQN algorithm and the number of games significantly appear to be positively correlated, as indicated by this graphical representation.

In contrast, there is a clear difference when comparing this graph to the one for the typical DQN algorithm. For an equivalent number of games, the Double DQN method routinely achieves an average score higher than the DQN algorithm. This striking difference points to the Double DQN algorithm performing better in terms of average score results. The data essentially suggests that the Double DQN method produces noticeably higher average scores than the DQN algorithm when the number of games grows. Thus, it can be concluded that the Double DQN method performs better than the normal DQN algorithm when it comes to average score accomplishment, demonstrating its effectiveness and superiority in agent performance optimization.

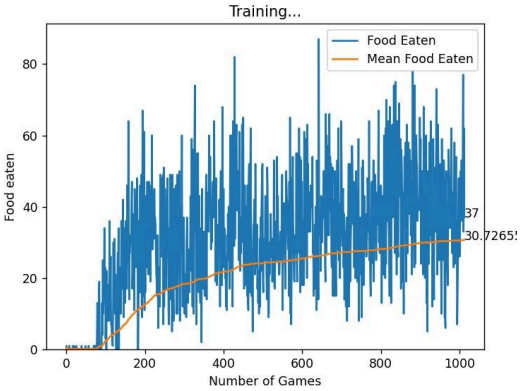


Fig. 9: Apples per episode for Double DQN algorithm

The orange line in Figure 9, which shows the "Mean Food Eaten" for every game, has an increasing trajectory that indicates a noticeable increase in the average amount of food consumed throughout the course of the observation period. Starting at a relatively low value, the line increases gradually but steadily, suggesting a gradual but constant increase in the amount of food consumed as the number of games increases. This increasing trend could suggest that performance or general effectiveness have improved in response to the games being played. Over the course of the gaming sessions, the mean food intake shows a progressive increase, which could indicate a good evolution due to improved skill, efficiency, or strategic prowess.

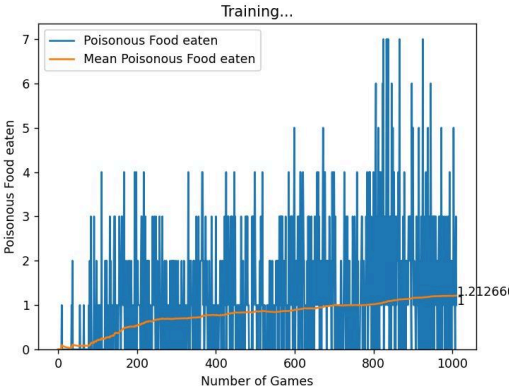


Fig. 10: Poisonous Apples per episode for Double DQN algorithm

Illustrated in Figure 10 is the agent's average consumption of poisonous food, depicted by the orange line. This graph not only highlights an escalating trend in the consumption of poisonous items as the games progress but also establishes a connection between this increase and the growing length of the agent within the game. As discussed earlier, the agent strategically consumes poisonous apples as a trade-off to avoid more severe penalties, especially when facing the risk of colliding with obstacles or itself. The progressive rise in the consumption of poisonous food aligns with its heightened length.

Table 1: Average Score Per Episode for Each Method

| Episode | DQN | Double DQN | Dueling DQN |
|---|---|---|---|
| 200 | 14.5 | 15.0 | 15.1 |
| 400 | 24.2 | 21.9 | 22.1 |
| 600 | 25.6 | 25.8 | 26.1 |
| 800 | 26.8 | 27.8 | 28.3 |
| 1000 | 28.6 | 29.6 | 30.6 |

While the results depicted in Table 1 showcase impressive performances across all three algorithms—DQN, DDQN, and Dueling DQN—it's noteworthy that even with the scores achieved, far surpassing human capability, the Dueling DQN algorithm consistently outperforms its counterparts. Despite the baseline success of DQN and the improvements seen with DDQN, Dueling DQN manages to exhibit superior performance, attaining higher average scores in each of the specified training sessions. This underscores the efficacy of

the Dueling DQN approach in pushing the boundaries of the agent's learning capabilities beyond what is achieved by the other algorithms, highlighting its exceptional ability to optimize reward gains.

Table 2: Average Poisonous Apples Eaten Per Episode for Each Method

| Episode | DQN | Double DQN | Dueling DQN |
|---|---|---|---|
| 200 | 0.49 | 0.65 | 0.61 |
| 400 | 0.64 | 0.71 | 0.76 |
| 600 | 0.83 | 0.96 | 0.88 |
| 800 | 1.01 | 0.97 | 1.08 |
| 1000 | 1.09 | 0.98 | 1.21 |

We observed a notable pattern: although the number of poisonous apples eaten remained relatively low, the length of the snake increased exponentially over episodes. This suggests that the models effectively learned to avoid or minimize the consumption of poisonous apples, allowing the snake to increase the reward successfully. The exponential growth in snake length and the apples eaten indicates a successful balance between navigation efficiency and the avoidance of harmful elements in the game.

Concluding the analysis with an in-game visual, all methods, side by side:

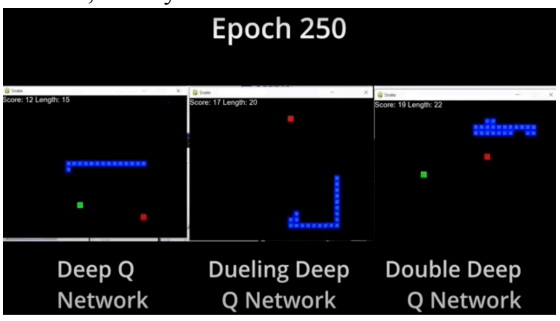


Fig. 11: Epoch 250 for each algorithm

In this context, "Epoch 250" means that the AI models have gone through 250 training episodes or iterations. An epoch in machine learning is commonly a complete pass over the entire training dataset, but in the case of reinforcement learning, it often refers to a single episode or game. During each epoch, the AI has a chance to learn from its actions and improve its strategy for playing the game. By the 250th epoch, the models had numerous opportunities to optimize their policies based on the rewards they've accumulated from the game environment.

## V. Conclusion

In this project, the objective was to assess the adaptability of various reinforcement learning models in the context of a modified Snake game. The chosen models included a Deep-Q Network (DQN), a Double Deep-Q Network (DDQN), and a Dueling Deep-Q Network. The game environment was established using the Pygame library, with modifications such as the introduction of poisonous food to add complexity.

The literature review drew insights from key papers, such as "Playing Atari with Deep Reinforcement Learning" by Mnih et al., "Deep Reinforcement Learning with Double Q-learning" by van Hasselt et al., "Dueling Network Architectures for Deep Reinforcement Learning" by Wang et al., and "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Hessel et al. These papers provided foundational concepts and modifications that influenced the implementation of the chosen models.

The methodology outlined the game environment, reward system, introduction of poisonous food, and the neural network models for the Q algorithms. Notably, due to performance considerations, a linear neural network was used instead of a CNN. The Deep Q-Network, Double Deep Q-Network, and Dueling Deep Q-Network algorithms were re-implemented, incorporating modifications suggested in the literature.

The results and analysis section presented visual representations of the performance of each agent, including reward values, apple counts, and timesteps. The addition of poisonous food was found to introduce complexity, requiring agents to balance pursuing rewards and avoiding pitfalls strategically. Results indicated improvements over time in terms of average scores and poisonous and non-poisonous foods eaten for the implemented algorithms.

Table 1 summarizes the average score per episode for each method, showing the progression of scores over training sessions. Dueling DQN exhibited the highest average scores, suggesting its effectiveness in achieving the project's primary objective.

In conclusion, the project successfully re-implemented and evaluated different reinforcement learning models in a modified Snake game environment. The findings contribute to understanding the strengths and weaknesses of each approach, paving the way for further improvements

and potential applications of these models in gaming scenarios.

### *A. Discussion*

Our study suggests that both DQN, Double DQN and Dueling DQN are suitable for our "Snake" game. As the results suggest, Dueling DQN is the most effective method among all methods.

DQN and Double DQN both achieved commendable maximum scores of close to 80, whereas Dueling DQN surpassed this limit, reaching an impressive 83. However, what sets Dueling DQN apart is not only its superior peak performance but also its higher mean score, showcasing a consistently better overall performance. This indicates that, on average, Dueling DQN not only attains higher individual scores but also demonstrates greater consistency in achieving favorable outcomes across multiple episodes, emphasizing its efficacy in our modified Snake game environment.

Potential reasons for the observed performance differences among the DQN, Double DQN, and Dueling DQN mainly include value decomposition, adaptability to complex environments, efficient exploration, and mitigating overestimation bias. Dueling DQN explicitly decouples the estimation of the state value function and the advantage function. This separation allows the agent to better understand the value of different actions in each state. In the Snake game, where strategic decision-making is crucial, this decomposition provides a more refined understanding of the consequences of different actions. The dueling architecture is designed to handle complex environments where certain actions may have different effects on the overall state value. In the Snake game with a poison apple system and the potential threat of the snake trapping itself on dead-ends, the ability of Dueling DQN to assess the value of actions independently contributes to superior adaptability and decision-making. Also, Dueling DQN might exhibit more efficient exploration due to its explicit handling of state values and advantages. This could be particularly beneficial in navigating the Snake game environment, leading to better learning and decision-making. These are the main reasons found as to why Dueling DQN performed the best. Following Dueling DQN, we have found Double DQN performed the second best. Double DQN addressed the issue of overestimation bias present in standard DQN. While this helps in mitigating overestimation, it may not offer the same level of refinement in action value estimates as the explicit separation provided by Dueling DQN. Finally, as outlined before, the original DQN probably suffers from overestimation bias. These overestimations could lead to suboptimal decision-making, as the agent may not accurately assess the consequences of its actions and potentially cause dead-ends.

It is evident that Double Deep Q-Network excels in our modified Snake game due to its innovative architecture, distinctly segregating the value and advantage functions. This architectural choice enhances the agent's ability to efficiently estimate the inherent value of each state, fostering a refined balance between exploration and exploitation while minimizing overestimation bias. The advantages of Double DQN suggest that its unique framework enables more effective learning and decision-making.

### *B. Further Improvements*

To further optimize Dueling DQN's performance, a multi-faceted approach can be employed, encompassing meticulous fine-tuning of hyperparameters, exploration of diverse network architectures, utilization of ensemble methods for robust predictions, integration of prioritized experience replay for informative learning, adaptive learning rate scheduling for dynamic adjustments, and experimentation with reward shaping strategies. These enhancements aim not only to amplify Dueling DQN's existing strengths but also to unravel nuanced insights into its superior adaptability and potential avenues for continuous refinement in our dynamic gaming environment.

As proposed in the Rainbow Deep RL paper, exploration of ensemble methods can enhance the performance of our methods. Combining the knowledge from diverse agents can lead to a more robust and adaptable decision-making process in the Snake game.

One other avenue for enhancing the performance of our methods might be adopting an adaptive discount factor. As the agent learns to survive for longer its own body length becomes its number one concern at the later stages of the game. By adopting an adaptive discount factor, where the body length corresponds with a lower discount factor, we can encourage the agent to take their time before rushing to the reward. This approach potentially might encourage the agent to stack its body in more favorable positions so that it won't put itself on dead-ends.

Finally, introducing variability into the initialization process by using different random seeds can significantly impact the training and performance of our methods. Due to long training times, we opted not to try out different random seeds. This approach might promote robust learning, allowing the agent to adapt to various initial states and encouraging the acquisition of more generalized strategies that are effective across different scenarios. Additionally, employing different seeds mitigates the risk of overfitting to a specific set of initial conditions.

## REFERENCES


[1]. Volodymyr Mnih et al., “Playing Atari with Deep Reinforcement Learning,” arXiv (Cornell University), Dec. 2013, doi: https://doi.org/10.48550/arxiv.1312.5602.

[2]. H Van Hasselt, “Deep reinforcement learning with double q-learning,” *Cornell University,* Sept, 2015.

[3]. ]Z. Wang, T. Schaul, M. Hessel, Hado van Hasselt, M. Lanctot, and Nando de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” *arXiv (Cornell University)*, Nov. 2015, doi: https://doi.org/10.48550/arxiv.1511.06581.

[4]. M. Hessel *et al.*, “Rainbow: Combining Improvements in Deep Reinforcement Learning,” *arXiv (Cornell University)*, Oct. 2017, doi: https://doi.org/10.48550/arxiv.1710.02298.

[5]. Zhizhou Ren, “On the Estimation Bias in Double Q-Learning,” *Department of Computer Science, University of Illinois at Urbana-Champaign*, Jan. 14, 2022.

[6]. M. S. Ausin, “Introduction to Reinforcement Learning. Part 4. Double DQN and Dueling DQN,” *Medium*, Nov. 25, 2020. https://markelsanz14.medium.com/introduction-to-reinforcement-learning-part-4-double-dqn-and-dueling-dqn-b349c9a61ea1