Single and Multi-Agent Reinforcement Learning in Double Snake

CS329 RL Project

November 23, 2025

Abstract

This project implements and evaluates various Multi-Agent Reinforcement Learning (MARL) algorithms in a grid-based Snake environment. We explore two distinct modes: a **Competitive** zero-sum scenario using Nash Q-Learning and Deep Nash Q-Learning, and a **Cooperative** scenario utilizing DQN, Double DQN (DDQN), SARSA, Monte Carlo, and Tabular Q-Learning. We analyze the stability, convergence, and win/survival rates of these agents. Results indicate that Deep Nash Q-Learning successfully approximates equilibrium strategies in high-dimensional state spaces, while DDQN provides the most stable performance in the cooperative setting with an average evaluation score of 12.89.

1 Introduction

Reinforcement Learning (RL) in multi-agent environments presents unique challenges, particularly the non-stationarity of the environment as agents update their policies simultaneously. This project applies RL to a two-player Snake game. The project is divided into two domains:

1. **Competitive:** Agents compete for food, where one agent's gain is the other's loss.
2. **Cooperative:** The Agent, controlling both the snakes, attempts to survive and maximise score within the same grid, navigating around each other.

2 Environment and Problem Formulation

2.1 State Space

The environment represents the game state differently for the two modes to optimize learning:

- **Competitive Mode:** 2 agents, each having a feature vector of size 18. It includes danger detection (straight, left, right), current direction, relative food direction, relative opponent head direction, and a length comparison flag.
- **Cooperative Mode:** 1 agent with a feature vector of size 22. It includes expanded danger detection that accounts for the specific location of the other agent's body segments to prevent collisions. Controls both the snakes.

2.2 Action Space

The agents operate in a discrete action space $A = \{0, 1, 2\}$, representing relative direction changes:

- 0: Continue Straight
- 1: Turn Left

- 2: Turn Right

In the Deep Nash implementations, the network outputs joint action values ($3 \times 3 = 9$ pairs) to compute the equilibrium.

2.3 Reward Structure

- **Food Reward:** +10 for eating.
- **Death Penalty:** -10 for collision with walls, self, or opponent.
- **Survival:** Small negative reward (-0.1) or 0 to encourage active food seeking.
- **Opponent Death (Competitive):** +5 bonus if the opponent dies.

3 Methodology

3.1 Competitive Algorithms

3.1.1 Nash Q-Learning

We extended tabular Q-learning to a general-sum game. At each step, instead of maximizing individual Q-values, the agent solves a Linear Programming (LP) problem to find the Nash Equilibrium of the stage game defined by the joint Q-values $Q(s, a^1, a^2)$. The 'scipy.optimize.linprog' library is used to solve for the maxmin strategy.

3.1.2 Deep Nash Q-Learning

To handle the continuous nature of the game flow and larger effective state spaces, we implemented a Deep Neural Network (DNN) to approximate the Q-values.

- **Architecture:** Fully connected layers (Input $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 9).
- **Training:** A replay buffer of capacity 50,000 is used. The target value is calculated based on the Nash value of the next state's Q-matrix.

3.2 Cooperative/Single-Agent Algorithms

We benchmarked several standard algorithms, each tailored to address specific challenges in cooperative multi-agent reinforcement learning:

- **SARSA (On-Policy):** This tabular method updates the Q-value based on the current action and the next action taken by the agent. It is on-policy, meaning it evaluates and improves the policy that is used to make decisions. SARSA is suitable for environments where the agent needs to learn from its current behavior.
- **Q-Learning (Off-Policy):** Unlike SARSA, Q-Learning is off-policy and updates the Q-value based on the maximum possible reward from the next state, regardless of the agent's actual next action. This makes it more exploratory and often faster to converge in deterministic environments.
- **First-Visit Monte Carlo:** This method estimates the value of a state-action pair by averaging the returns of the first visit to that pair in an episode. It is particularly useful in episodic tasks where the agent can learn from complete trajectories.

- **Deep Q-Network (DQN):** Extends Q-Learning by using a neural network to approximate the Q-values, enabling it to handle high-dimensional state spaces. DQN employs experience replay and a target network to stabilize training.
- **Double DQN (DDQN):** An improvement over DQN, DDQN decouples the action selection from the Q-value evaluation to reduce overestimation bias. This results in more stable and reliable learning, especially in environments with noisy rewards.

4 Experiments and Results

4.1 Hyperparameters

Table 1 summarizes the configuration used during the final training runs.

Parameter	Competitive (Deep Nash)	Cooperative (DQN/DDQN)
Episodes	3,000 - 5,000	20,000
Learning Rate (α)	0.001	0.001
Discount (γ)	0.95	0.99
Batch Size	64	64
Buffer Size	50,000	20,000
Epsilon Decay	0.995	0.995
Hidden Layers	128	256

Table 1: Hyperparameters for Deep Learning models.

4.2 Competitive Analysis

Training logs for Nash Q-Learning and Deep Nash Q-Learning indicate a high prevalence of "Draws".

- **Deep Nash Q Results (100 episodes):**
  - Agent 1 Wins: 3%
  - Agent 2 Wins: 97%
  - Draws: 97.0%
- **Tabular Nash Q Results (100 episodes):**
  - Agent 1 Wins: 15.0%
  - Agent 2 Wins: 50.0%
  - Draws: 35.0%

The high draw rate suggests the Deep Nash Q strategy in this specific environment tends towards conservative survival rather than aggressive killing, likely due to the high penalty for mutual death. The lower draw rate in Nash Q stratergy could be due to the fact that the tabular method learns exact state-action values and therefore converges more cleanly to decisive, exploitative strategies, whereas deep Nash Q generalizes conservatively and over-penalizes risky aggression. The tabular agents thus more readily identify profitable attack opportunities and engage in high-stakes interactions during training, resulting in both wins and losses, whereas the deep agents drift towards mutually safe, non-engaging behavior that prioritizes survival and thus produces disproportionately many draws.

4.3 Cooperative Analysis

The Deep Learning approaches significantly outperformed tabular methods due to the state space complexity.

- **DQN Performance:** Achieved an average score of 15.89 with a maximum score of 35 in evaluation.
- **DDQN Performance:** Achieved an average score of 12.89 with a maximum of 37. While the average was slightly lower than DQN, training logs suggest DDQN exhibited more stable loss convergence.

4.4 Comparative Analysis

The competitive and cooperative approaches in this project highlight distinct challenges and outcomes in multi-agent reinforcement learning:

- **Objectives:** Competitive algorithms aim to maximize individual rewards in a zero-sum setting, often leading to conservative strategies to avoid mutual losses. Cooperative algorithms, on the other hand, focus on maximizing collective rewards, requiring agents to balance individual gains with team objectives.
- **Challenges:** Competitive algorithms, particularly Nash Q-Learning, face computational challenges in solving Nash equilibria at each state, which can be prohibitive in real-time scenarios. Cooperative algorithms, especially deep learning-based methods like DQN and DDQN, struggle with stability and convergence in high-dimensional state spaces but benefit from experience replay and target networks.
- **Outcomes:** Competitive strategies often result in high draw rates, reflecting a tendency towards survival rather than aggressive play. Cooperative strategies, particularly DDQN, demonstrate robust performance with higher average scores and more stable training dynamics.
- **Scalability:** Deep learning-based approaches in both domains show better scalability to larger state spaces compared to tabular methods, making them more suitable for complex environments.

4.5 Evaluation Results

Table 2 summarizes the evaluation results for all models, including both competitive and cooperative algorithms. Metrics such as average score, maximum score, and average episode length are reported.

Algorithm	**Avg. Score**	**Max Score**	**Min Score**	**Avg. Episode Length**
Deep Nash Q-Learning	0.03 ± 0.17	0	0	655.0 ± 113.6
Nash Q-Learning	3.67 ± 2.83	15	0	92.1 ± 75.4
Double DQN (DDQN)	12.89 ± 7.49	37	1	130.38 ± 78.08
DQN	15.89 ± 7.52	35	0	164.74 ± 79.13
Monte Carlo (On-Policy)	6.15 ± 2.74	13	0	77.50 ± 37.88
Q-Learning	8.09 ± 3.66	17	1	135.20 ± 77.87
SARSA (On-Policy)	1.86 ± 1.79	7	0	285.32 ± 165.97

Table 2: Evaluation results for all models. Metrics include average score, maximum score, minimum score, and average episode length.

5 Supporting Figures

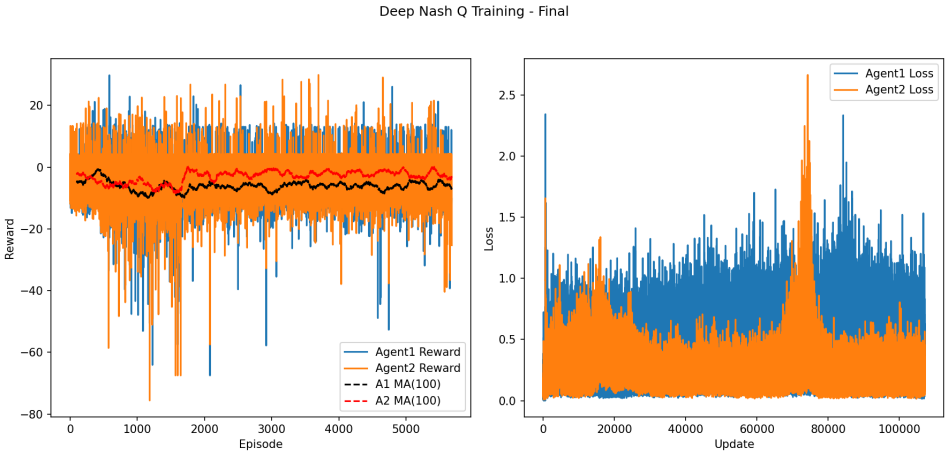

Figure 1: Deep Nash Q-Learning (Final loss (Avg.): **0.291**)

Figure 2: Tabular Nash Q-Learning

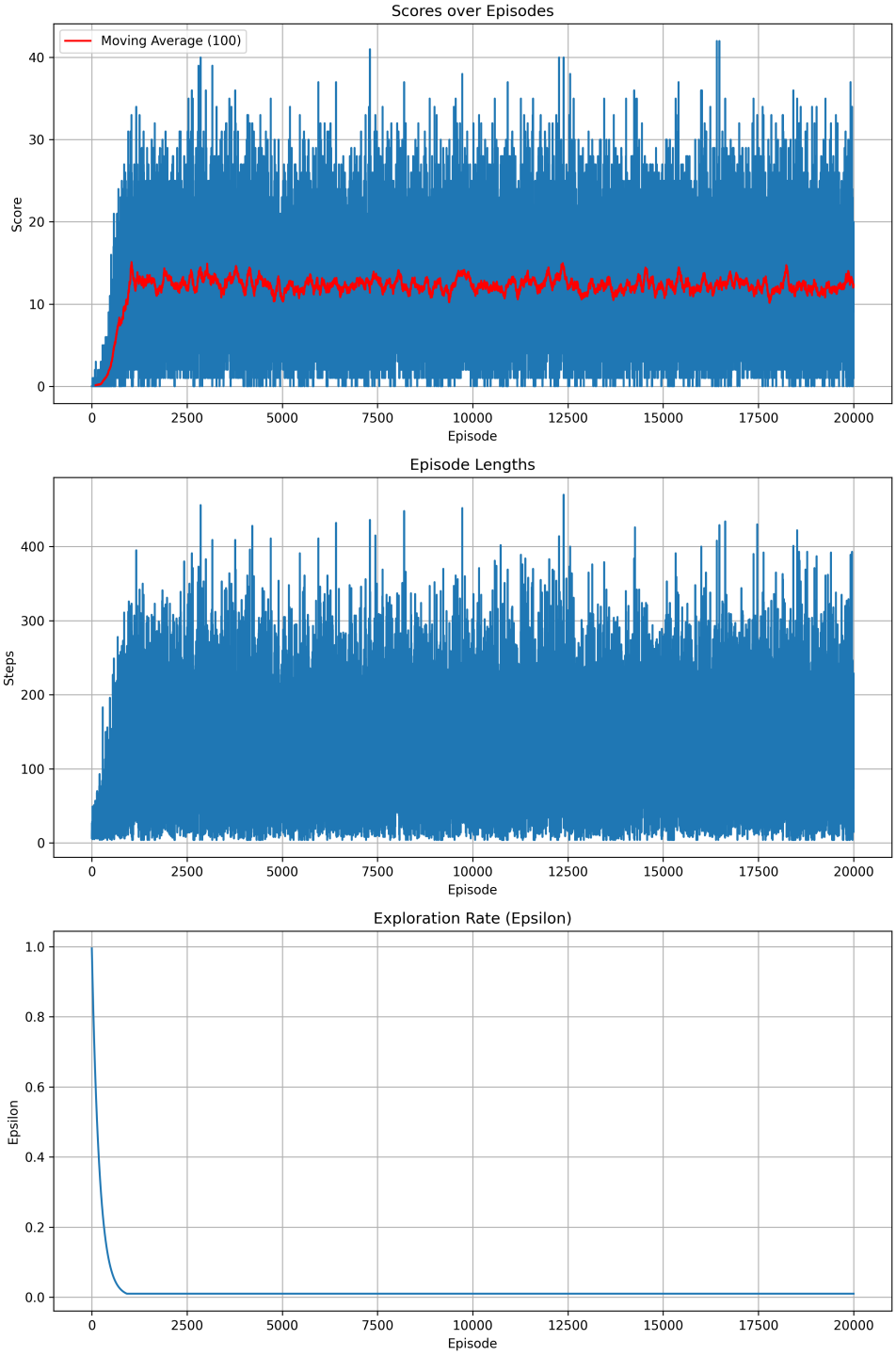

Figure 3: Cooperative DQN

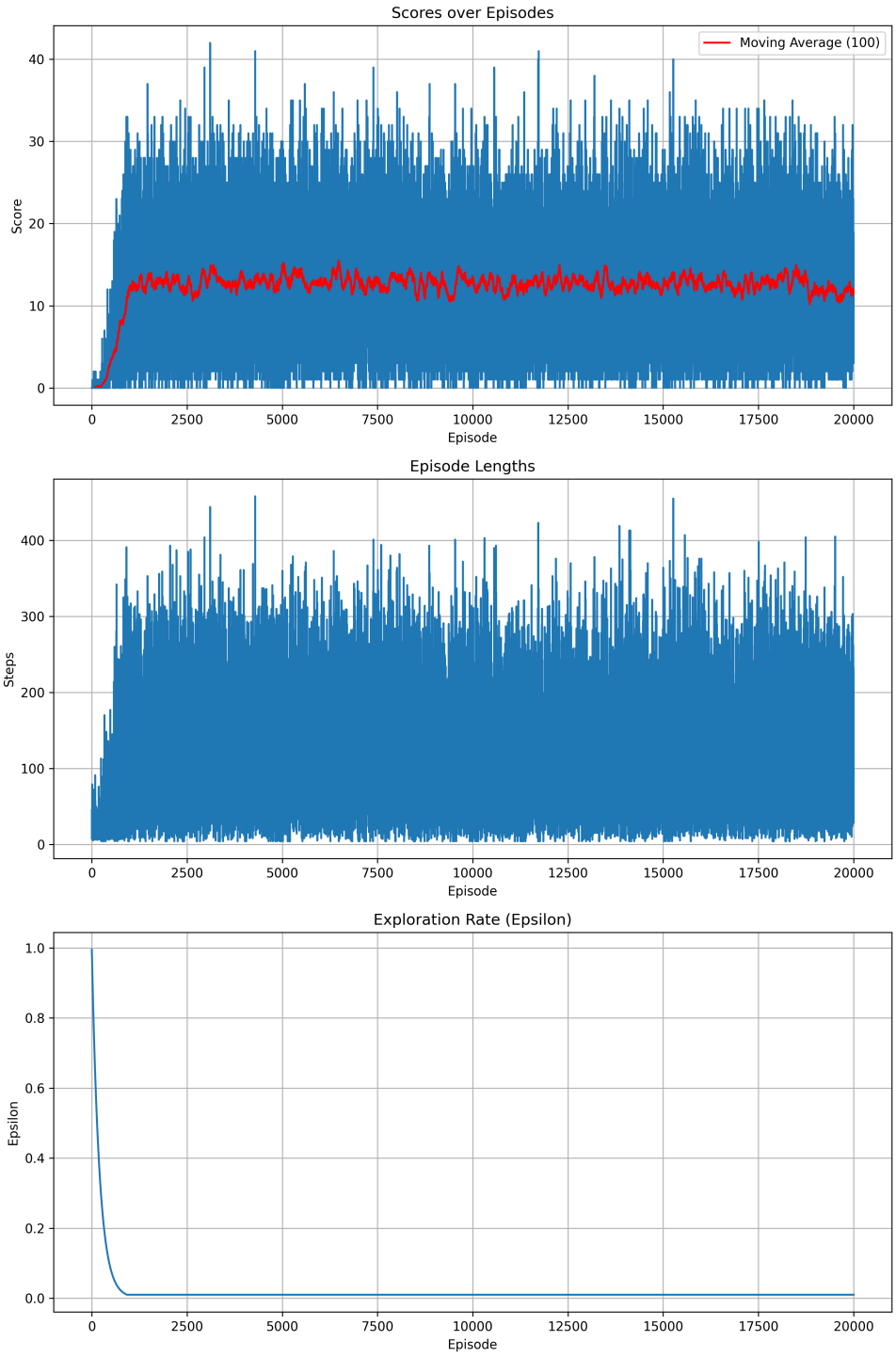

Figure 4: Cooperative DDQN

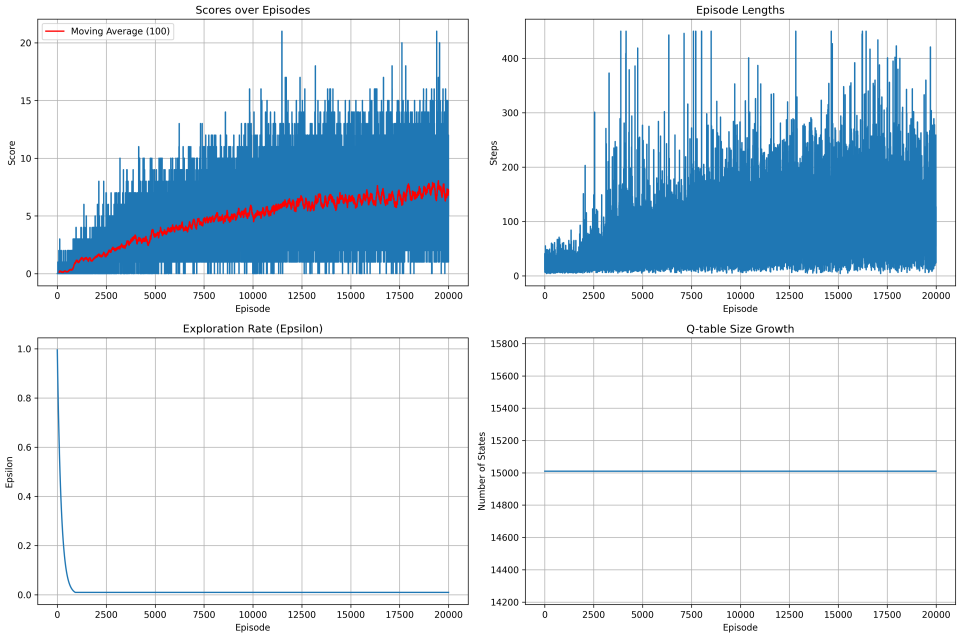

Figure 5: Tabular Q-Learning

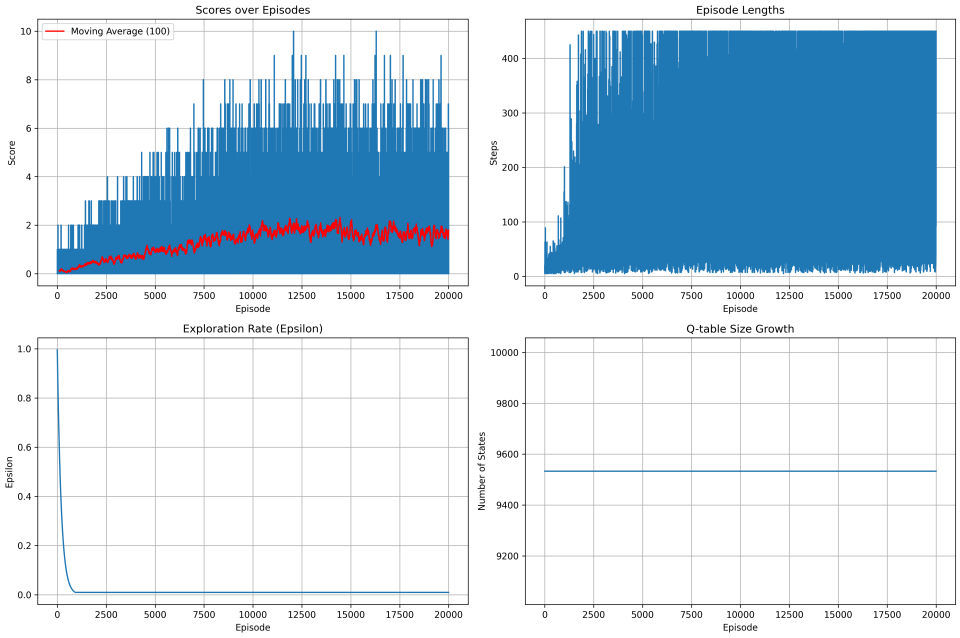

Figure 6: SARSA (On-Policy)

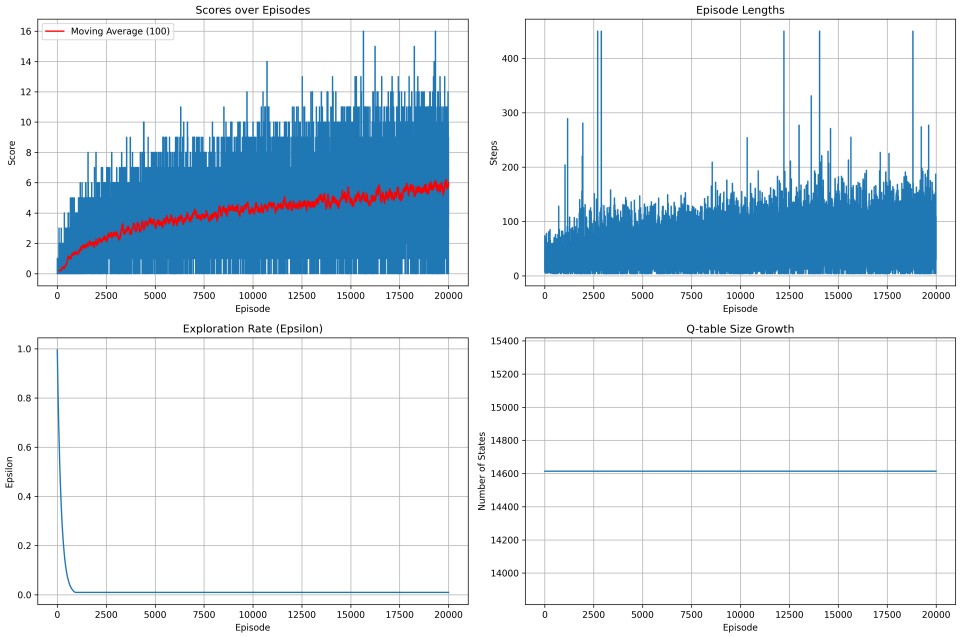

Figure 7: Monte Carlo (On-Policy)

6 Contributions

1. Aarsh Wankar: Deep Q and double deep q and results for corporative. Also organized the repo :D.

2. Abhinav Khot: Monte Carlo and SARSA algorithm implementation. Also organized the repo ;).

3. Jaskirat Singh Maskeen: Training Models, Competitive results, Organising the cluttered repository :), and Report.

4. Karan Sagar Gandhi: Implemented Nash Q, Deep Nash Q and Q learning algorithms. Also implemented the Game and rendering logic for the setup.

7 Conclusion

This project demonstrated the application of MARL to Snake. In the competitive domain, Nash Q-Learning successfully identified safe strategies, resulting in prolonged survival (draws) rather than quick eliminations. In the cooperative domain, Deep Q-Learning methods proved robust, with DQN and DDQN agents learning to navigate the grid and consume food effectively while avoiding dynamic obstacles (the other agent).

Future work could involve implementing Minimax-Q for strictly zero-sum scenarios or adding communication channels between cooperative agents to improve collision avoidance in tighter grid spaces.