Multi-Agent Reinforcement Learning in 2 Snake: Competitive and Cooperative Approaches

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:** Agents attempt to survive and maximize 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:** 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:** 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.

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:

- **Tabular Methods:** SARSA (On-Policy), Q-Learning (Off-Policy), and First-Visit Monte Carlo.
- **Deep Q-Network (DQN):** Uses an MLP (256 hidden units) with an Experience Replay buffer.
- **Double DQN (DDQN):** Decouples action selection from evaluation to reduce maximization bias.

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 (200 episodes):**
  - Agent 1 Wins: 5.0%
  - Agent 2 Wins: 10.0%
  - Draws: 85.0%
- **Tabular Nash Q Results:**
  - Agent 1 Wins: 5.0%
  - Agent 2 Wins: 15.0%
  - Draws: 80.0%

The high draw rate suggests the Nash equilibrium strategy in this specific environment tends towards conservative survival rather than aggressive killing, likely due to the high penalty for mutual death.

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.

5 Supporting Figures

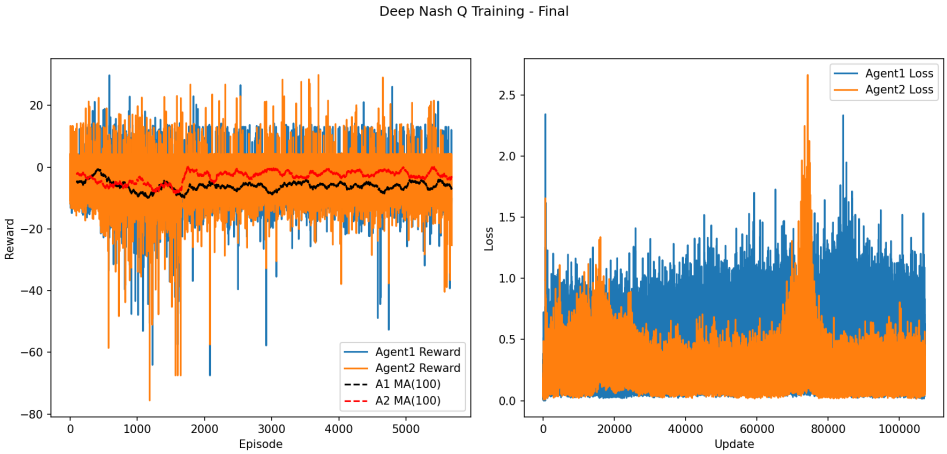

Figure 1: Deep Nash Q-Learning (Final loss (Avg.): **0.291**)

Figure 2: Tabular Nash Q-Learning

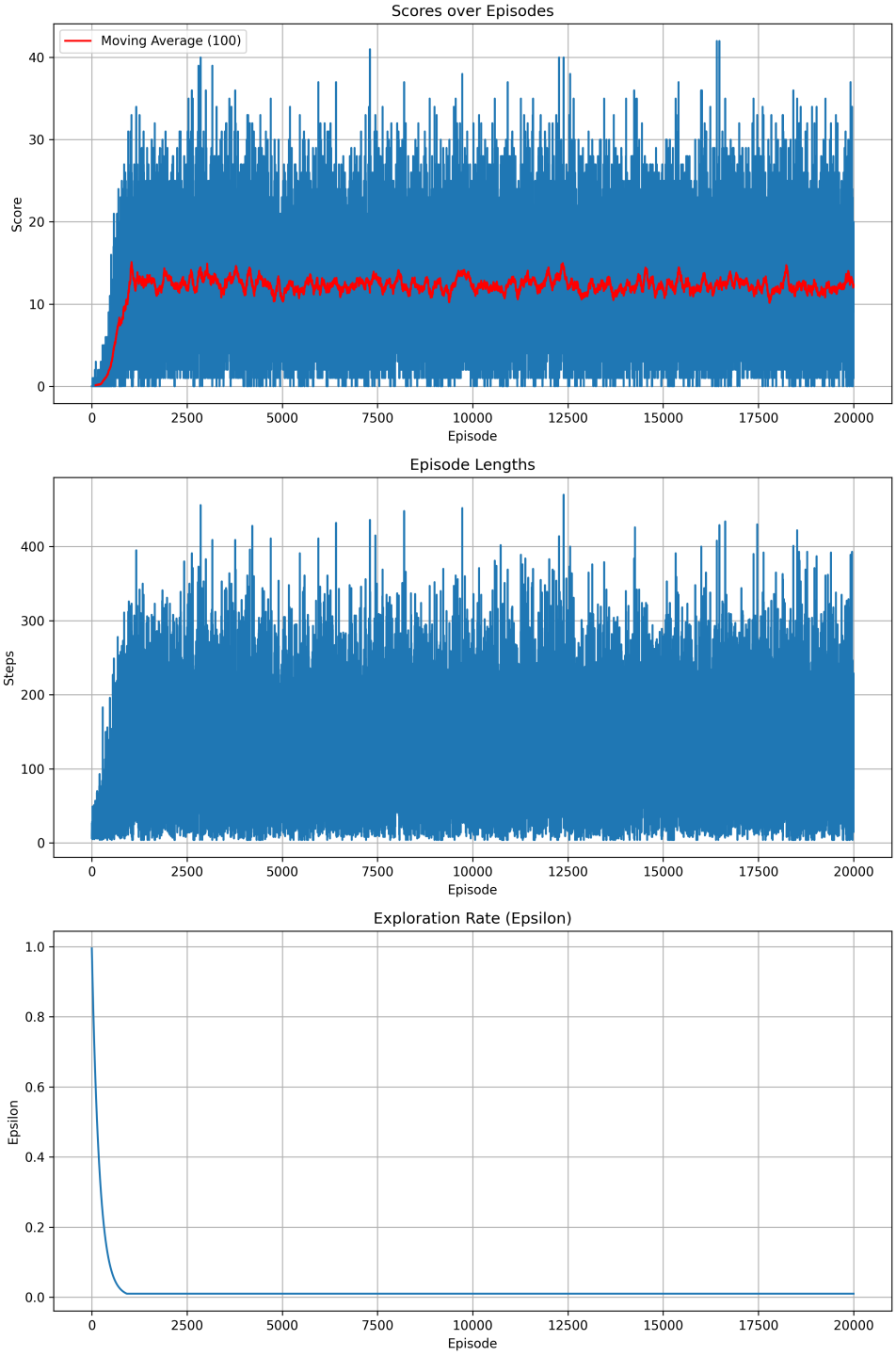

Figure 3: Cooperative DQN

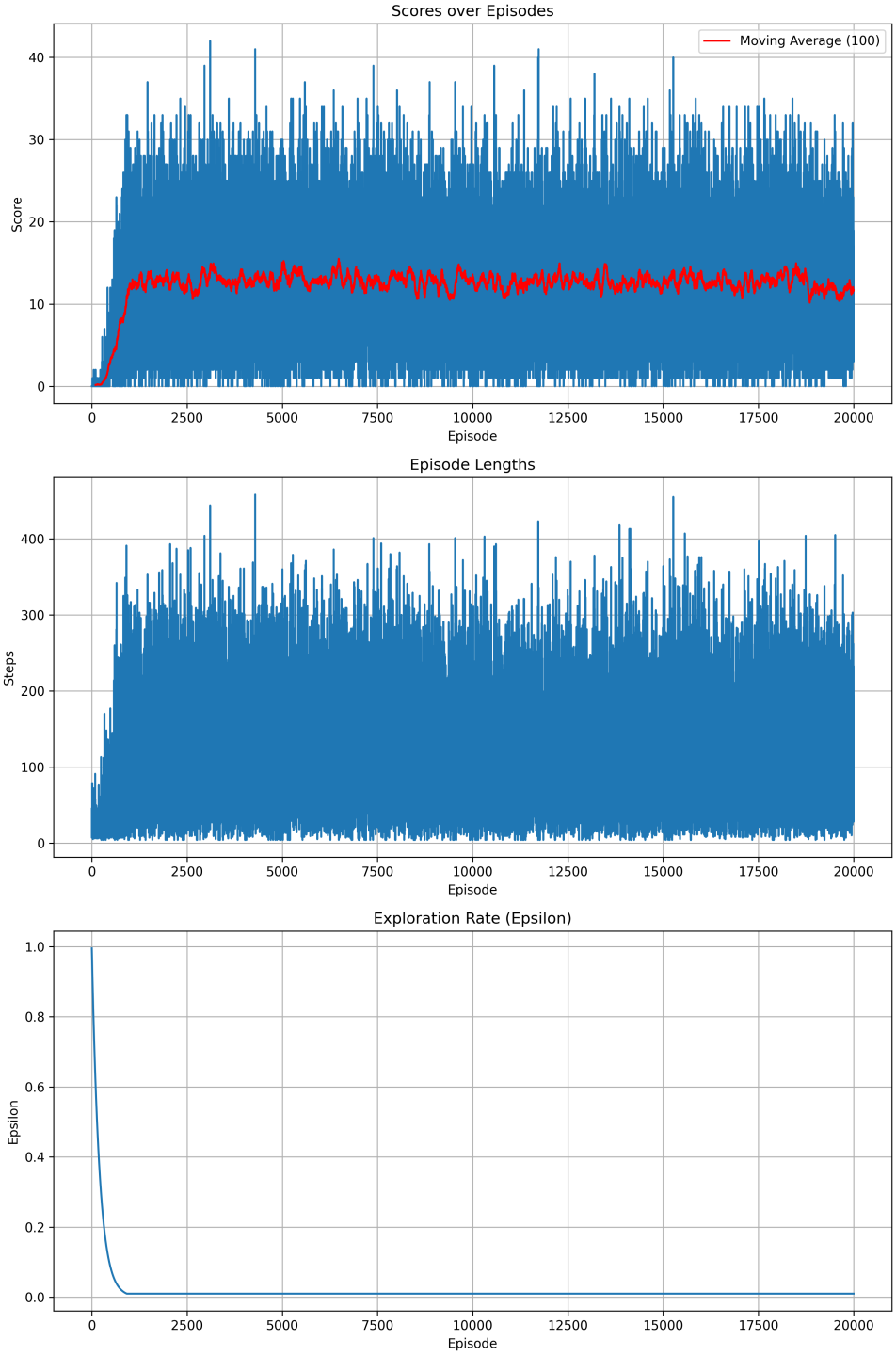

Figure 4: Cooperative DDQN

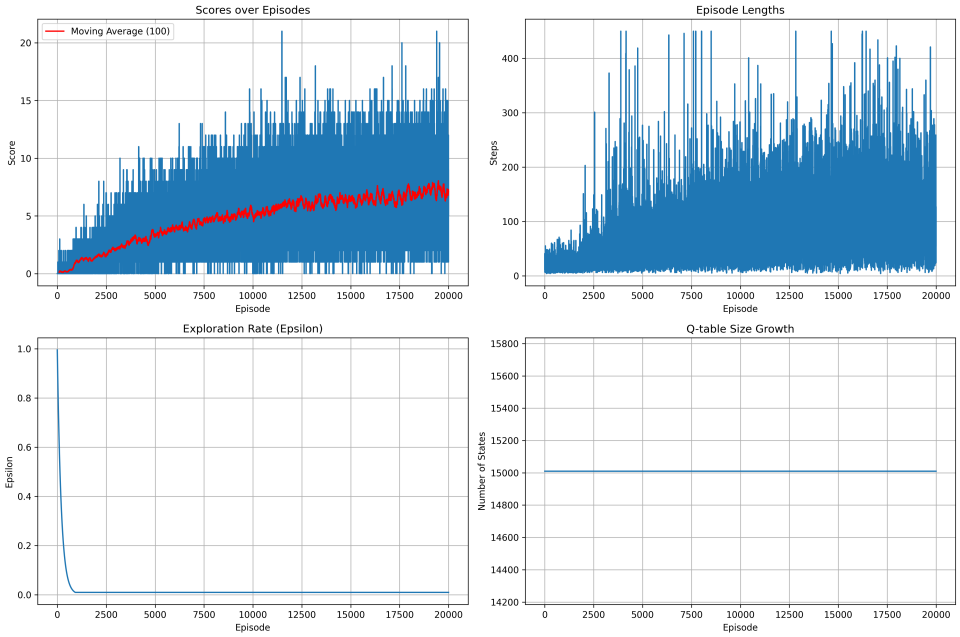

Figure 5: Tabular Q-Learning

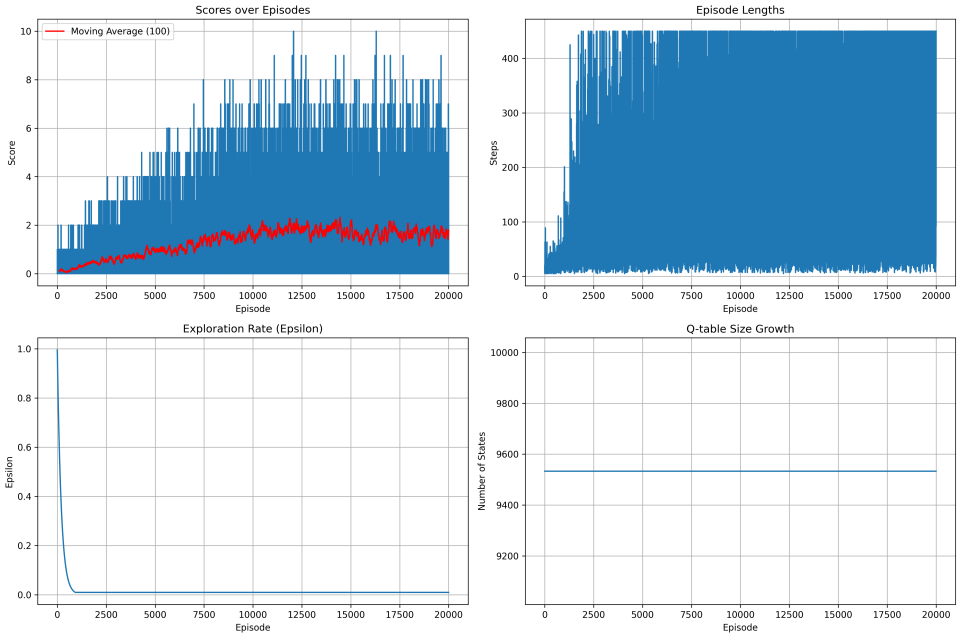

Figure 6: SARSA (On-Policy)

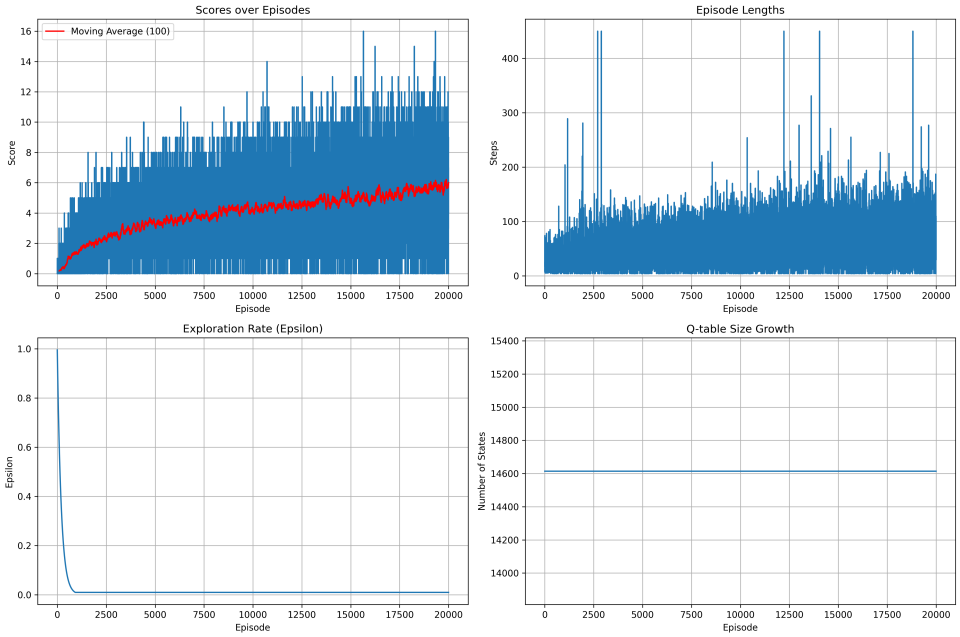

Figure 7: Monte Carlo (On-Policy)

6 Contributions

1. Karan Gandhi: Implemented Nash Q, Deep Nash Q and Q learning algorithms
2. Jaskirat Singh Maskeen: Training Models, Competitive results, Organising the cluttered repository, and Report.
3. Aarsh Wankar: Deep Q and double deep q and results for corporative.
4. Abhinav Khot: Monte Carlo and SARSA algorithm implementation.

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.