CSC 580 AI II | HW4 | Name: Om Prakash Gunja | Student ID: 2131025

Q learning for Snake Game

## Introduction

The goal was to analyze and optimize decision making of agent through reinforcement learning and improve its performance over time may be through tuning hyper params.

My approach:

* Analyze current params’s performance against different outputs and epsilons
* Balancing Exploration vs. Exploitation: Finding the right epsilon value to ensure the agent explores enough without being random.
* Refining Learning Efficiency: Adjusting alpha (learning rate) to optimize how quickly the agent learns from mistakes.
* Prioritizing Long-Term Rewards: Fine-tuning gamma (discount factor) to encourage forward-thinking decisions.
* Evaluating Performance: Comparing different Q-tables to see how training progresses over iterations.

## Agent Modifications and Justifications

#### 1. Q-Table Storage

The Q-table is stored as a dictionary with integer-encoded states using state\_to\_int(), reducing memory usage and enabling quick lookups and updates.

#### 2. Action Selection

The select\_action() function implements an epsilon-greedy strategy, initializing unseen states and choosing either a random action (explore) or the best-known action (exploit) via select\_greedy().

def select\_action(self, state):

""" Epsilon-greedy action selection. """

if np.random.rand() < self.epsilon:

return np.random.choice(self.action\_space) # Explore

else:

return self.select\_greedy(state) # Exploit

#### 3. Q-Table Updates

Before updating Q-values, update\_Qtable() ensures both current and next states exist. It also handles terminal states properly to prevent unnecessary updates and instability.

def update\_Qtable(self, state, action, reward, next\_state, done):

""" Update the Q-table using the Q-learning algorithm. """

state\_int = self.state\_to\_int(state)

next\_state\_int = self.state\_to\_int(next\_state)

self.visited.add(state\_int)

# Ensure states exist in Q-table

self.init\_state(state)

self.init\_state(next\_state)

# Terminal state handling

max\_next\_q = 0 if done else max(self.Q[next\_state\_int].values())

# Q-learning update rule

self.Q[state\_int][action] += self.alpha \* (

reward + self.gamma \* max\_next\_q - self.Q[state\_int][action]

)

# Decay epsilon

self.adjust\_epsilon()

#### 4. Unique State Tracking

A visited set tracks unique states encountered. The num\_states\_visited() function provides a measure of exploration efficiency, helping analyze agent coverage.

#### 5. Q-Tables

The write\_qtable() and read\_qtable() functions enable saving and loading Q-values, ensuring training progress is retained and allowing hyperparameter tuning without restarting.

## Experiment Analysis and Evaluation

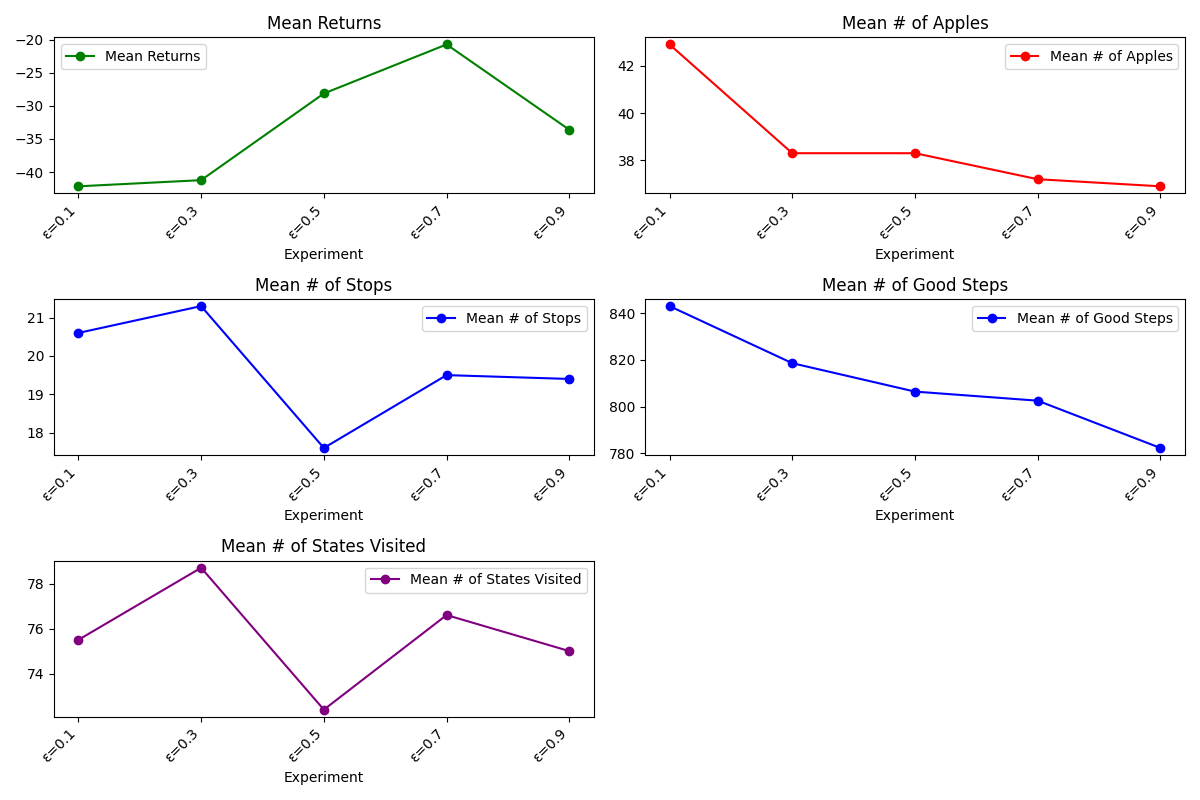
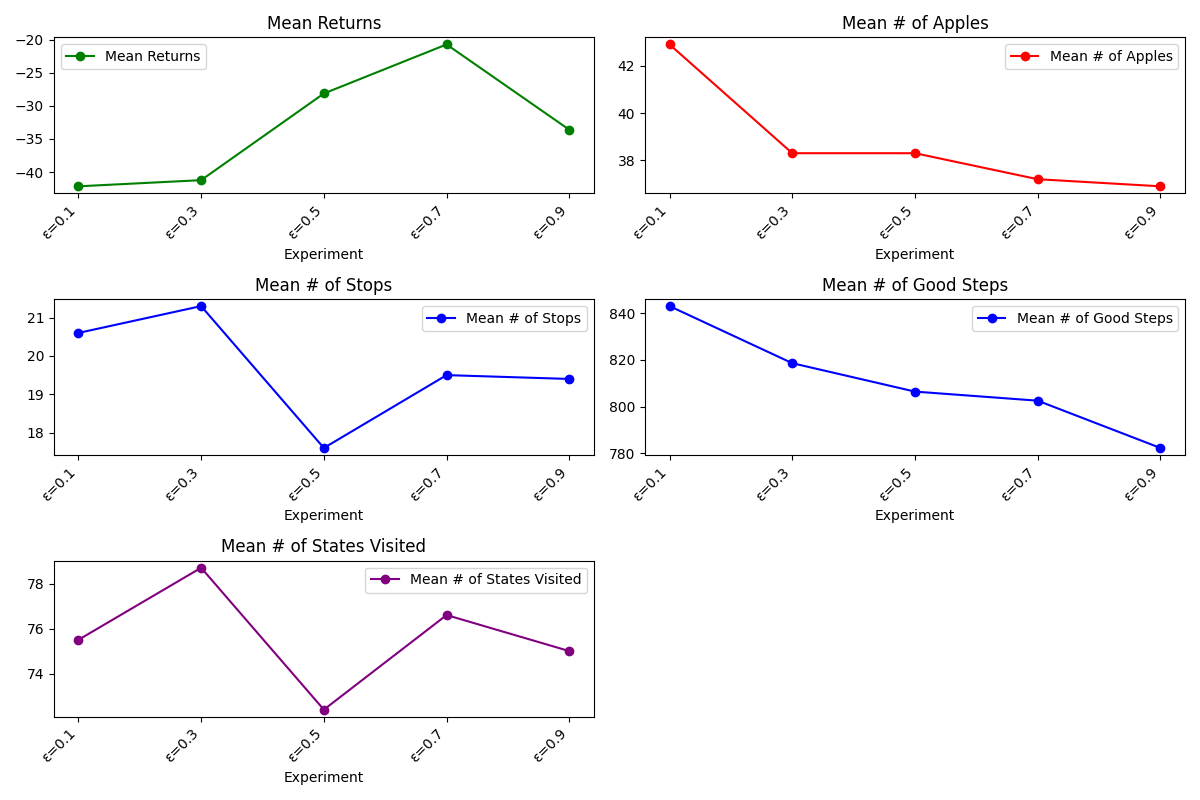
To optimize the Q-learning agent, I conducted a structured evaluation by tuning epsilon, alpha, and gamma. The goal was to find the best hyperparameters that improve performance metrics such as mean returns, apples collected, stops, good steps, and state exploration.

## Step-by-Step Analysis

#### Epsilon Tuning (best\_epsilon\_qtable.png)

* Evaluated multiple epsilon values (0.1 to 0.9) to balance exploration vs. exploitation.

#### Observations:

* Lower epsilon (0.1 - 0.3) led to better exploitation but limited exploration.
* Higher epsilon (0.7 - 0.9) encouraged exploration but resulted in more suboptimal decisions.
* Best epsilon (0.7) achieved the highest mean returns and stable learning behavior.

#### Initial vs. Best Epsilon Comparison(comparison\_epsilon.png)pasted-movie.png

* Compared performance between initial training and best epsilon.

#### Observations:

* Mean returns increased, indicating better overall performance.
* Stops decreased, meaning fewer collisions and more stable movement.
* Apples collected slightly decreased, suggesting the agent prioritized safe movement over aggressive food collection.

#### Alpha & Gamma Tuning (best\_alpha\_gamma\_qtable.png)

#### Evaluated different combinations of alpha and gamma to improve learning efficiency.pasted-movie.png

#### Observations:

* Lower alpha (0.5) resulted in slower learning, while higher alpha (0.9) led to unstable updates.
* Higher gamma (0.99) encouraged long-term planning but sometimes caused over-prioritization of future rewards.
* Best combination (alpha= 0.7, gamma = 0.85) got balanced performance, improving stability and learning speed.

#### Final Comparison (comparison\_all.png)pasted-movie.png

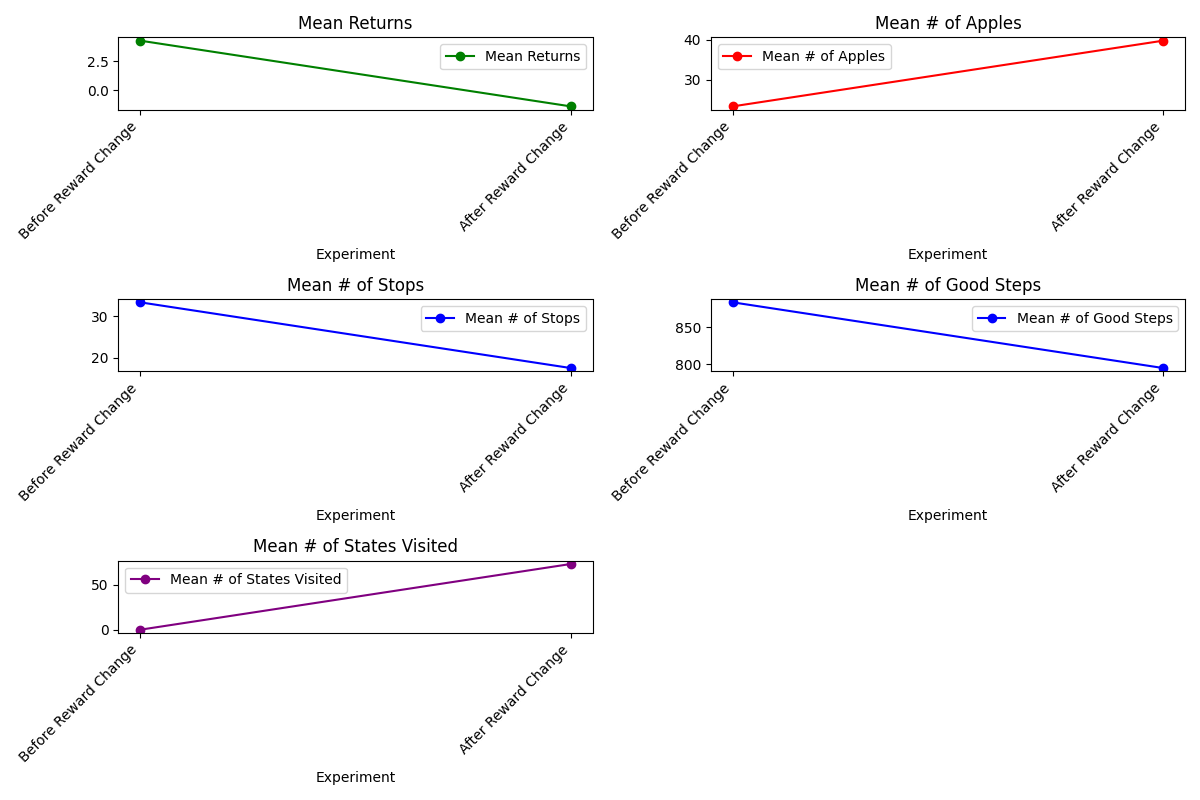
* Compared Initial Q-table, Best Epsilon Q-table, and Tuned Q-table.

Key Trends:

* + Mean returns improved significantly after tuning both epsilon and alpha - gamma.
  + Stops decreased, confirming improved movement stability.
  + Good steps and state exploration stabilized, indicating efficient decision making.

#### Key Observations & Takeaways

* Best-performing model: Tuned Q-table with epsilon = 0.7, alpha = 0.7, gamma = 0.85.
* Tuning epsilon first helped stabilize training, and further alpha-gamma tuning optimized learning speed and decision efficiency.
* Unexpected Findings: Higher epsilon (above 0.7) led to random behavior instead of better exploration, suggesting diminishing returns on excessive exploration.

While observing the agent’s performance, I noticed that most terminations occurred due to the snake colliding with its own body. This motivated me to experiment with the reward function by increasing the penalty for self-collisions( -200). The hypothesis was that by discouraging tail collisions, the agent would navigate more efficiently and focus on collecting apples. This experiment used epsilon = 0.7 to maintain a good balance between exploration and exploitation.

#### Key Findings (reward\_change\_comparison.png)

Apples Collected Increased More apples were collected after increasing the penalty for self-collisions.

Stops Decreased : The agent moved more smoothly and avoided unnecessary halts.

States Visited Increased : Higher exploration led to more diverse state visits.

Mean Returns Decreased : Slightly Increased exploration led to riskier decisions.

Good Steps Decreased : More early terminations due to over-exploration.

Future improvs:

Fine tuning epsilon decay , experimenting with reward scaling , training with longer episodes and implementing DQN would improve performance

#### Reflections on the Assignment

* Hands-on Learning: Implementing Q-learning in a dynamic environment like Snake reinforced theoretical concepts through practical application.
* Importance of Hyperparameter Tuning: Adjusting epsilon, alpha, and gamma proved crucial in optimizing agent performance.
* Challenges Faced: Debugging unexpected behaviors, balancing exploration-exploitation, and ensuring smooth learning convergence were significant hurdles.
* Iterative Approach: Step-by-step tuning and experimentation were necessary to refine the agent’s decision-making process.