# Assignment II

**Machine Learning I**
Data Science and Engineering

**May 2024**


Salma Alejandra Caisaguano Barzallo - 100496504
Raquel Jaén Delgado - 100496615

**Universidad Carlos III de Madrid**

# Contents

**1 Phase I: Selection of the State Information and Reward Function** **2**

1.1 State Description . . . . . 2

1.2 Reward Function Design . . . . . 2

**2 Phase II: Generation of the Agent** **4**

2.1 Implementation of Missing Functions . . . . . 4

2.2 Agent's functionality . . . . . 5

2.3 Parameter Tuning . . . . . 5

2.4 Incremental Development . . . . . 7

2.5 Result and Analysis . . . . . 7

**3 Phase III: Enhancement of the Agent** **8**

3.1 Enhanced Decision-Making . . . . . 8

3.2 Technical Implementation . . . . . 8

3.2.1 State Representation . . . . . 8

3.2.2 Reward Function Modifications . . . . . 8

3.2.3 Q-Learning Adjustments . . . . . 8

3.3 Testing and Evaluation . . . . . 9

3.4 Parameter Tuning . . . . . 9

3.5 Results and Analysis . . . . . 11

**4 Conclusions** **12**

## Introduction

In this assignment, you will apply the Q-learning algorithm in the domain of the Snake game, to build an agent that works automatically, chases the food in the board and avoids it s own body. The objective of the agent will be to maximize the score obtained. To do this, we will use reinforcement learning techniques.

# 1 Phase I: Selection of the State Information and Reward Function

## 1.1 State Description

The way we describe the current situation in the game is crucial for the agent. It gives the agent all the information it needs to decide what to do next. When we chose how to describe the state, we thought about a few things to make sure the agent can make good decisions no matter what's happening:

- **Locating the Food in Relation to the Snake:** We analyzed the food's position compared to the snake's head, both horizontally and vertically. This insight guides the agent on the best path to reach the nearest food source.

- **Measuring the Distance to the Food:** We broke down the distance to the food into handy categories like 'CLOSE' (1), 'MEDIUM' (2), and 'FAR' (3). This aids the agent in gauging whether the food is nearby or requires a bit more journeying.

These approaches were selected for their versatility across varying game board sizes. By doing so, our agent can navigate intelligently regardless of the board's dimensions.

## 1.2 Reward Function Design

Creating a good reward system is really important because it helps the agent learn how to play the game better and aim for higher scores. We followed these basic ideas:

- **Rewarding Good Actions:** We want to encourage actions that lead to good outcomes, like eating food or getting closer to it. By giving rewards for these actions, the agent learns to do things that help it win.

- **Punishing Bad Actions:** On the other hand, we also want to discourage actions that lead to bad results, like crashing into walls. So, we give negative rewards for these actions, teaching the agent to avoid them.

- **Finding the Right Balance:** We need to make sure the reward system encourages the agent to try new things to figure out the best strategies, but also to stick with what it knows works well. This way, it keeps learning and improving as it plays more games.

These concepts were put into practice during the agent's development to ensure its efficacy in navigating the game environment.

## 2 Phase II: Generation of the Agent

Now, we could focus on implementing a Q-learning-based agent to play the Snake game. Our objective was to develop an agent capable of learning optimal strategies through interactions with the game environment.

### 2.1 Implementation of Missing Functions

We started by implementing the missing functions required for Q-learning, including *calculate_reward*, *get_state*, and *update_q_table*. Additionally, we completed the main loop in `SnakeGame.py` to orchestrate the interaction between the agent and the game environment.

- **calculate_reward**: the agent assigns rewards to its actions based on how well they align with its goals. For instance, successfully grabbing a piece of food might earn a reward, while bumping into obstacles or itself could result in a less desirable outcome. By considering factors like distance to food and potential dangers, the agent learns to associate actions with different levels of reward, guiding its decision-making process.

- **get_state**: This function captures key details about the game state, such as where the snake and the food are located, and translates them into a format that the agent can understand. By discretizing this information into a structured representation, the agent gains insights into the current situation, helping it choose the best course of action.

- **update_q_table**: the agent calculates the value of its actions in different situations, taking into account both immediate rewards and expected future rewards. By continuously refining its estimates of action values, the agent gradually improves its decision-making abilities, becoming more adept at navigating the game environment.

- **Main loop**: the main loop guides the agent's adventure in the Snake game. It starts a new episode where the agent watches the game, makes choices, and affects what happens. The loop helps the agent learn from what happens and change how it plays. With each episode, the loop helps the agent get better at the game.

### 2.2 Agent's functionality

The agent's functionality was based on Tutorial 4, where we built and updated a Q-table according to the Q-learning algorithm. We discretized the game state based on the relative position of the food with respect to the snake and its distance. This discretization allowed us to construct a Q-table with appropriate dimensions to capture the state-action space.

### 2.3 Parameter Tuning

In the exploration and exploitation phase of our Q-learning based Snake game agent, we focused on fine-tuning three critical parameters: the learning rate ($\alpha$), the discount factor ($\gamma$), and the exploration rate ($\varepsilon$). These parameters are pivotal in balancing the trade-off between exploring new actions and exploiting known strategies to maximize the agent's performance.

#### Learning Rate ($\alpha$)

The learning rate $\alpha$ influences the extent to which new information affects the existing knowledge. A higher $\alpha$ allows the agent to adjust quickly to new data, whereas a lower $\alpha$ makes learning more stable but slower. We tested values at 0.1, 0.3, and 0.5 to observe how different rates of learning would impact the agent's ability to converge to an optimal policy.

#### Discount Factor ($\gamma$)

The discount factor $\gamma$ determines the importance of future rewards. A $\gamma$ close to 1 makes future rewards more significant, promoting strategies that benefit in the long term rather than immediate gains. We maintained $\gamma$ at 0.9, consistent with strategies that value long-term gains, crucial for the Snake game where long-term survival is key.

#### Exploration Rate ($\varepsilon$)

The exploration rate $\varepsilon$ affects the agent's likelihood of trying a random action versus following the strategy suggested by the Q-table. Initially high rates encourage exploration of the state space, while lower rates help in honing the strategy as the agent learns. We experimented with $\varepsilon$ values of 0.3, 0.75, and 0.9 to test various degrees of exploration.

These parameters were iteratively adjusted in a series of tests, each involving 1000 episodes, to identify the combination that offers the best balance between exploration and exploitation, essential

for optimal performance in varying game scenarios.

| $\alpha$ | $\gamma$ | $\varepsilon$ | Max reward | Min reward | Mean reward | Eaten Food |
|---|---|---|---|---|---|---|
| 0.1 | 0.9 | 0.3 | 8205 | −4304 | 484.236 | 84 |
| 0.1 | 0.9 | 0.75 | 6591 | -3688 | -185.641 | 27 |
| 0.1 | 0.9 | 0.9 | 5035 | -3256 | -306.728 | 24 |
| 0.3 | 0.9 | 0.3 | 8158 | -5088 | 425.729 | 91 |
| 0.3 | 0.9 | 0.75 | 4643 | -3920 | -322.982 | 24 |
| 0.3 | 0.9 | 0.9 | 4849 | -3432 | -387.274 | 16 |
| 0.5 | 0.9 | 0.3 | 7586 | -2888 | 473.432 | 98 |
| 0.5 | 0.9 | 0.75 | 5143 | -4232 | -267.656 | 26 |
| 0.5 | 0.9 | 0.9 | 5103 | -3912 | -353.835 | 19 |

**Table 1:** Parameter Tuning and Resulting Metrics

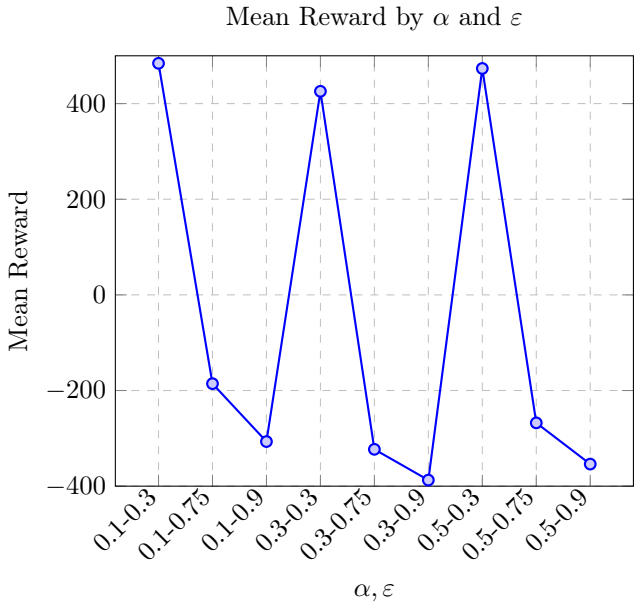


**Figure 1:** Plot of mean rewards for different parameter settings

The following observations were made regarding the tuning of learning rate ($\alpha$), discount factor ($\gamma$), and exploration rate ($\varepsilon$) for our Q-learning based Snake game agent:

- **Impact of Learning Rate ($\alpha$)**: When the learning rate ($\alpha$) is higher, like 0.5, the game player learns faster and does better. This is especially important in games like Snake where things change quickly.

- **Role of Exploration Rate ($\varepsilon$)**: How much the player explores makes a big difference. When the exploration rate ($\varepsilon$) is lower, like 0.3, the player learns best. This means once the player

knows a lot, it's better to use what it knows instead of trying random stuff.

- **Consequences of High Exploration**: But when the player explores too much, like with an exploration rate of 0.9, it does worse. This is because it ends up doing a lot of not-so-good moves, which makes it harder to learn the best strategies.

- **Stability and Performance Trade-offs**: The best results come from finding a balance between trying new things and using what's already learned. So, having a medium exploration rate and a higher learning rate works best. This means the player learns well without trying too many random moves.

### 2.4 Incremental Development

Following the recommended approach, we adopted an incremental development process. We started with smaller game boards (150x150) to build and refine the agent's behavior. Subsequently, we scaled up to larger board sizes, such as (400x400), ensuring that the agent performed well across different maze configurations.

### 2.5 Result and Analysis

The agent demonstrated the ability to navigate the Snake game environment effectively. It learned to maximize rewards by avoiding collisions, consuming food, and optimizing its movement strategy.

Tuning the hyperparameters $\alpha$, $\varepsilon$, and $\gamma$ significantly influenced the agent's learning dynamics. We observed that moderate exploration ($\varepsilon$) combined with a suitable learning rate ($\alpha$) and discount factor ($\gamma$) resulted in stable and efficient learning.

We evaluated the agent's performance by tracking key metrics such as total reward per episode, maximum and minimum rewards, and mean reward over multiple episodes. Additionally, we validated the Q-table updates by inspecting the file storing the table and verifying that experiences led to the correct updates.

# 3 Phase III: Enhancement of the Agent

As we progress into Phase III of our Machine Learning assignment, our focus shifts towards refining the capabilities of our agent to handle diverse scenarios within the Snake game environment. Our objective now is to enhance the agent's decision-making process by introducing considerations for the snake's body within the game state.

We will integrate the necessary changes into the existing codebase, building upon the functionalities developed in previous phases.

## 3.1 Enhanced Decision-Making

In this phase, our goal was to make the snake smarter about avoiding its own body. We added new information to the snake's decision-making process, helping it to see where its body is and avoid running into itself.

## 3.2 Technical Implementation

### 3.2.1 State Representation

We changed how we describe the state of the game; now, the snake knows where the closest part of its body is in relation to the food which helps the snake avoid parts of its body while it tries to get food.

### 3.2.2 Reward Function Modifications

We improved how we reward or penalize the snake to help it make better choices. Now, the snake gets points for moving towards food, which encourages it to go after its target. If the snake moves towards its own body or away from the food, it loses points. This teaches the snake to move carefully and avoid danger. Also, if the snake hits a wall or itself, it loses a lot of points, which helps it learn to stay safe while moving around the game area.

### 3.2.3 Q-Learning Adjustments

We made some changes to the Q-learning method to handle the new and more complex ways we describe the snake's world. We have included more types of situations the snake can learn from, allowing it to come up with strategies for a wider variety of challenges. We also adjusted how quickly

the snake learns and how often it tries new moves. This helps the snake get better at finding food while avoiding making the same mistakes, ensuring it learns well and keeps improving.

### 3.3 Testing and Evaluation

In the testing phase, we put the snake in a bigger playing area to give it more room and challenges. This setup let us run fewer but more intense learning sessions, which helped because the game got harder. We kept a close eye on how the snake performed, noting how much food it ate and how often it managed to not run into itself. This information was very important for checking if the new changes we made were working well.

### 3.4 Parameter Tuning

| $\alpha$ | $\gamma$ | $\varepsilon$ | Max reward | Min reward | Mean reward | Eaten Food |
|---|---|---|---|---|---|---|
| 0.1 | 0.9 | 0.3 | 28616 | -2994 | 2387.129 | 461 |
| 0.1 | 0.9 | 0.75 | 11796 | -4472 | 168.252 | 124 |
| 0.1 | 0.9 | 0.9 | 6193 | -4528 | -394.258 | 50 |
| 0.3 | 0.9 | 0.3 | 29213 | -4624 | 1865.072 | 333 |
| 0.3 | 0.9 | 0.75 | 9252 | -4440 | -73.168 | 96 |
| 0.3 | 0.9 | 0.9 | 6413 | -4704 | -439.839 | 21 |
| 0.5 | 0.9 | 0.3 | 23381 | -2568 | 1575.833 | 311 |
| 0.5 | 0.9 | 0.75 | 10198 | -4584 | -121.544 | 60 |
| 0.5 | 0.9 | 0.9 | 5973 | -4520 | -363.071 | 29 |

**Table 2:** Parameter Tuning and Resulting Metrics

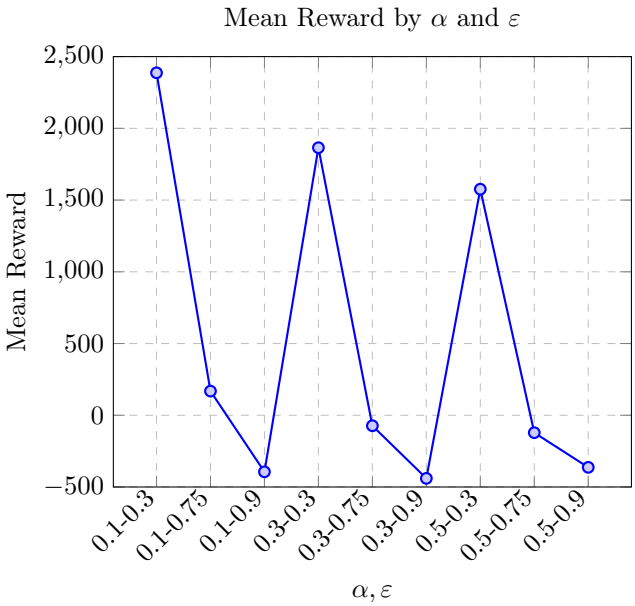


**Figure 2:** Plot of mean rewards for different parameter settings in Phase III

In Phase III, where we upgraded the agent's decision-making to help it avoid crashing into its own body, we noticed some important things about adjusting the parameters:

- **Learning Rate ($\alpha$) Effectiveness**: A higher learning rate, like $\alpha = 0.5$, helped the snake adapt faster to changes in the game. This was really useful for teaching the snake to move around its own body quickly, especially as it got longer.

- **Exploration Rate ($\varepsilon$) Dynamics**: Just like before, having a lower exploration rate ($\varepsilon = 0.3$) led to better performance. It let the snake use its well-learned strategies effectively. On the other hand, a high exploration rate ($\varepsilon = 0.9$) didn't work well because it made the snake choose not-so-good actions, which was bad, especially in the more complicated game.

- **Adjusting to Tougher Game Rules**: When we changed how the agent made decisions to include avoiding its own body, it got better at dodging itself. This was super important for playing longer and getting higher scores.

- **Finding the Right Settings**: We found that having a good balance in the learning parameters was key. Using a middle-ground exploration rate with a higher learning rate ($\alpha$) worked best. It helped the snake explore new strategies while still using what it already knew well, making it learn fast and play better overall.

### 3.5 Results and Analysis

The updates we made to how the snake makes decisions have worked out well. The snake is now better at not bumping into itself while it tries to eat food. This shows that our new way of defining the game and the changes to the scoring system are helping the snake be more careful and make smarter moves.

# 4 Conclusions

Throughout our project, we made a Q-learning-based agent for the Snake game, improving its abilities step by step. Here's what we achieved and learned:

First, we focused on describing the game state and creating a reward system. By considering the position and distance of the food relative to the snake, we gave the agent the necessary information to make good decisions. The reward system encouraged the agent to seek food and avoid crashing into walls, which helped it learn to play the game better.

Next, we built the agent using Q-learning. We implemented functions like *calculate_ reward*, *get_ state*, and *update_ q_ table*. These allowed the agent to learn from its actions and improve over time. We also adjusted parameters like the learning rate and exploration rate to find the best balance between trying new things and using what it already knew. This made the agent better at finding food and avoiding obstacles.

In the final phase, we made the agent smarter by teaching it to avoid its own body. We updated how we described the game state and changed the reward system to punish the agent for moving towards itself. This improvement helped the agent survive longer and get higher scores by avoiding self-collisions.

Overall, our step-by-step approach worked well. Each phase built on the previous one, making the agent more capable and effective. The agent showed it could learn to play the Snake game well, adapting to different game scenarios.