**Project Description: Snake Game (Author: Ilia Kabanov)**

My work on this project consisted of the following steps:

1. Formulating a request to ChatGPT to generate a game environment for Snake and studying the provided code.
2. Creating a template for the final code based on the response from ChatGPT, particularly designing a framework for implementing agent classes and a key function, start\_games, to run games with parallel training.
3. Implementing the Random\_Agent class.
4. Researching additional materials on Q-learning. Filling in the Q\_learning\_Agent class and analyzing errors by step-by-step debugging of the algorithm.
5. Testing the agent after analyzing errors and modifying the reward system.
6. Implementing the Logic\_Agent class based on the Q\_learning\_Agent. Designing logical heuristics for Snake's actions and testing the agent.
7. Adjusting the code to save key characteristics of the agents.
8. Comparing the agents based on their characteristics and creating histograms (e.g., scores per game).
9. Finalizing the code for better readability and clarity.
10. Writing the report.

Now, I would like to elaborate on each step of the process.

**Formulating the Request**

Formulating the request to ChatGPT did not take much time. However, more time was needed to understand its code and adapt it for my script.

**Agent Implementation**

Next, I considered possible implementations of agents. Besides a random agent, I thought about simple logical rules for Snake, such as moving toward food and avoiding obstacles. Additionally, I explored reinforcement learning methods like Q-learning and DeepQNetwork. Q-learning seemed simpler than DQN yet effective enough for a relatively simple game. Hence, I settled on three agents: random, "logical," and Q-learning.

The template for each agent included three functions: initialization, get\_next\_action, and get\_next\_location, as described in one of the analyses I reviewed.

**The** *start\_games* **Function**

The *start\_games* function launches a cycle of games for a selected agent. During each game, until the required number of games is completed:

* The initial positions of the Snake, food, and environment state are initialized.
* For each step, until the Snake "dies":
  + The agent selects and performs an action based on the current state.
  + The environment state is updated.
  + The Q-matrix is updated if the agent is in training mode.

The random agent chooses one of three movements randomly: forward, right, or left.

The implementation of environment states, parameters, and the Bellman function for Q-learning followed online sources (e.g., https://8thlight.com/insights/qlearning-teaching-ai-to-play-snake) and YouTube tutorials (https://www.youtube.com/watch?v=je0DdS0oIZk).

**Training and Debugging**

The training parameters and rewards initially did not allow the agent to learn effectively. To identify errors, I output the training process in text format, showing step-by-step Snake actions and learning progress. An example of game 500 out of 600 training games is in the file q\_learning\_agent\_game\_500.txt.

The issue was that the Snake was rarely rewarded—only for eating food—and frequent "explore" moves disrupted the learning process, causing it to forget rewards. Therefore, I introduced additional rewards: +1 for moving closer to food and -1 for moving away. I also minimized the random move coefficient to ensure the Snake retained rewards. After these adjustments, the Snake's learning improved significantly. Additionally, I increased the penalty for death from -10 to -100 to encourage avoiding obstacles. The reward for eating food remained +10, which is much smaller than the penalty for death. Lastly, I reduced the discount factor for future rewards because the Snake, seeing only one square ahead, misinterpreted threats, leading it to loop unnecessarily.

After these fixes, the Snake achieved an average score of 58 after 600 games, which is a good result. I'll explain why later.

**Logic Agent**

Rules for the "logical" agent were written directly into its Q-matrix, which was not updated during the game. For each environment state, the Snake acted according to pre-defined rules in the Q-matrix.

The following rewards were also specified: -100 for death, +10 for eating food, +1 for moving closer to food, and -1 for moving away.

On average, the Snake scored 57 points across 100 games.

**Comparison Metrics**

Apart from the total number of steps per game and steps to the first piece of food, I chose the following metrics for comparison:

* Game score (the most intuitive metric).
* Average number of steps to reach food.

A higher game score indicates a better agent. A lower average number of steps to food, all else being equal, suggests a more efficient agent.

Thus, these results were recorded during the games.

**Comparison with Other Works**

The author of the tutorial video (<https://www.youtube.com/watch?v=je0DdS0oIZk>) considers his results solid enough for a tutorial. According to his statistics, the average score of his agent was around 44, while my Logic\_Agent and Q\_learning\_Agent achieved an average score of 58.

Moreover, my Q-learning agent learned significantly faster, reaching this level after 600 games compared to 7,500 games in the tutorial video.

Both of my agents outperformed the random agent, whose average score was close to zero. A higher number of steps per game indicates better survival, which my two agents achieved. Additionally, they reached food faster both initially and throughout the game. These factors demonstrate that my two agents performed significantly better than the random agent and the Q-agent from the tutorial.

**Q-Agent vs. Logic Agent**

The Q-agent and Logic Agent showed very similar results, which is not surprising given their identical reward systems. However, the Q-agent occasionally makes random moves and accounts for future steps using the Bellman equation, unlike the Logic Agent. As a result, their performance slightly differs in the final metrics, though not by much.

**Reproducibility and Code Refinement**

For reproducibility, the random seed was set to 42.

At the end of the work, function descriptions and comments were added to the code for better clarity and readability.