Implementation of Q-Learning in the Mouse and Cat Game

Algorithm Presentation

This project implements the **Q-Learning** algorithm, a technique from **Reinforcement Learning**, to solve the classic **Mouse and Cat** problem within a maze environment. Q-Learning enables the agent (in this case, the mouse) to learn the optimal strategy to reach a goal (food) while avoiding capture by an adversary (cat). The algorithm relies on the iterative updating of a Q-table that stores the utility values for each state-action pair, allowing the mouse to make informed decisions over time.

Problem Description

Overview

The **Mouse and Cat** game is a strategic navigation problem set within a maze. The objective is for the mouse to reach the food while evading the cat, which seeks to capture it. The maze is represented as a two-dimensional grid where each cell can be a free space, a wall, or the location of the food. The challenge lies in navigating the maze efficiently, making strategic movements to outmaneuver the cat, and ultimately reaching the food without being caught.

Maze Configuration

The maze is defined by a matrix where each element represents a different type of cell:

- **0**: Free space where the mouse and cat can move.
- **1**: Wall that blocks movement.
- **2**: Food, the target location the mouse aims to reach.

Initial Setup

- **Mouse**: Starts at position (0, 0).
- **Cat**: Starts at position (10, 14), positioned closer to the mouse's goal to increase the challenge.
- **Food**: Located at position (4, 4).

The maze is designed to provide a balanced level of difficulty, ensuring that the mouse has a viable path to the food while the cat has sufficient opportunities to intercept or deter the mouse.

Objectives

- **Mouse**: Navigate from the starting position to the food efficiently while avoiding capture.
- **Cat**: Pursue the mouse with the objective of capturing it before it reaches the food.

Challenges

- **State Space Complexity**: The combined positions of the mouse and cat create a large state space, making it computationally intensive to explore all possible states.
- **Dynamic Adversary**: The cat moves with a certain probability advantage, allowing it to make additional moves, thereby increasing the difficulty for the mouse.
- **Balancing Exploration and Exploitation**: Ensuring that the mouse explores the environment sufficiently while also exploiting known strategies to maximize rewards.

Enhanced Maze and Movement Improvements

After extensive testing, the maze was expanded to a larger size to increase complexity and provide a more challenging environment for both the mouse and the cat. Additionally, the cat's movement logic was enhanced, allowing it to move randomly at a faster pace, increasing the challenge for the mouse. These enhancements aimed to create a more dynamic and realistic scenario where the mouse must employ strategic decision-making to navigate the maze effectively, balancing its efforts to reach the food while evading a more unpredictable and agile adversary.

How the Algorithm Solves the Problem

Theoretical Foundations

Q-Learning is a model-free Reinforcement Learning algorithm that enables an agent to learn the value of actions in particular states without requiring a model of the environment. The key components of Q-Learning include:

- **State (S)**: Represents the current configuration of the environment. In this scenario, the state is defined by the positions of both the mouse and the cat within the maze.
- **Action (A)**: The set of possible moves the mouse can take (up, down, left, right).
- **Reward (R)**: A numerical value received after performing an action. Rewards are structured to incentivize the mouse to reach the food and penalize it for invalid moves or proximity to the cat.
- **Q-Table (Q-table)**: A matrix that stores the utility values for each state-action pair. Initially, all values are set to zero.

Implementation Details

Initialization

The maze is defined as a NumPy array with designated positions for the mouse, cat, and food. The Q-Learning parameters are set as follows:

- **α (alpha)**: Learning rate (0.1), determining the extent to which new information overrides old information.
- **γ (gamma)**: Discount factor (0.95), valuing future rewards.
- **ε (epsilon)**: Exploration rate (1.0), which decays over time to balance exploration and exploitation.
- **Number of Episodes**: 10,000, ensuring comprehensive learning.
- **Maximum Steps per Episode**: 300, limiting the length of each training episode.
- **Cat's Extra Move Probability**: 0.3, giving the cat a chance to make an additional move, enhancing its advantage.

State Representation

Each state is uniquely encoded based on the positions of the mouse and the cat. The function state_to_index converts the combined positions into a unique index for the Q-table:

```
def state_to_index(mouse_pos, cat_pos):
    return (mouse_pos[0] * maze_width + mouse_pos[1]) * (maze_height * maze_width) + (cat_pos[0] * maze_width + cat_pos[1])
```

Action Selection: ε-Greedy Policy

At each step, the mouse selects an action based on the ε-greedy policy:

- **Exploration**: With probability ε, the mouse chooses a random valid action to explore new states.
- **Exploitation**: With probability 1 - ε, the mouse selects the action with the highest Q-value for the current state.

```
if random.uniform(0,1) < epsilon:
    possible = get_possible_actions(mouse_pos)
    if possible:
        action = random.choice(possible)
    else:
        action = random.randint(0, action_space_size -1)
else:
    action = np.argmax(Q_table[state_idx])
```

Q-Table Update

After performing an action, the Q-table is updated using the Q-Learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Where:

- s = current state
- a = action taken
- R = reward received
- s' = new state
- a' = possible actions from the new state

Cat's Movement Strategy

The cat employs a strategy to minimize the **Manhattan distance** to the mouse. The Manhattan distance between two points (x1, y1) and (x2, y2) is calculated as:

$$\text{Distance} = |x1 - x2| + |y1 - y2|$$

This distance metric ensures that the cat moves in a grid-like fashion towards the mouse, making strategic decisions to reduce the distance efficiently. Additionally, with a probability of 0.3, the cat can perform an extra move, increasing the likelihood of capturing the mouse or deterring its progress.

Training Process

The algorithm undergoes multiple episodes of training, where in each episode:

1. The mouse and cat start at their initial positions.
2. At each step:
    - The mouse selects an action based on the ε-greedy policy.
    - The mouse moves, and rewards are assigned based on the action's outcome.
    - The cat moves towards the mouse, possibly making an extra move.
    - The state transitions are recorded, and the Q-table is updated accordingly.

- The episode terminates if the mouse reaches the food, performs an invalid move, or is captured by the cat.

3. The exploration rate ε decays gradually to favor exploitation of learned strategies over time.

Through this iterative process, the mouse learns to navigate the maze effectively, balancing the exploration of new paths with the exploitation of known rewarding actions.

Results

Performance Metrics

The training progress is monitored through the total rewards accumulated in specific episodes, alongside the decayed exploration rate ε. Below are the observed results from various training milestones:

Episode 1000: Total Reward: -157, Epsilon: 0.3677
Episode 2000: Total Reward: -135, Epsilon: 0.1352
Episode 3000: Total Reward: -125, Epsilon: 0.0497
Episode 4000: Total Reward: -132, Epsilon: 0.0183
Episode 5000: Total Reward: -136, Epsilon: 0.0100
Episode 6000: Total Reward: -209, Epsilon: 0.0100
Episode 7000: Total Reward: -201, Epsilon: 0.0100
Episode 8000: Total Reward: 39, Epsilon: 0.0100
Episode 9000: Total Reward: 41, Epsilon: 0.0100
Episode 10000: Total Reward: -9, Epsilon: 0.0100

Interpretation of Results

- **Negative Rewards**: Indicate episodes where the mouse either failed to reach the food, made invalid moves, or was captured by the cat.
- **Positive Rewards**: Reflect successful episodes where the mouse reached the food, sometimes even while the cat posed a significant threat.
- **Exploration Rate (ε) Decay**: Demonstrates the transition from exploration to exploitation. Initially high, allowing the mouse to explore various actions, it decays over time, enabling the mouse to rely more on learned strategies.
- **Balanced Performance**: The alternating pattern of positive and negative rewards, especially in later episodes, indicates that the maze size and movement logic have been fine-tuned to create a balanced scenario. The mouse occasionally succeeds in reaching the food, while the cat effectively captures or deters it in other instances.

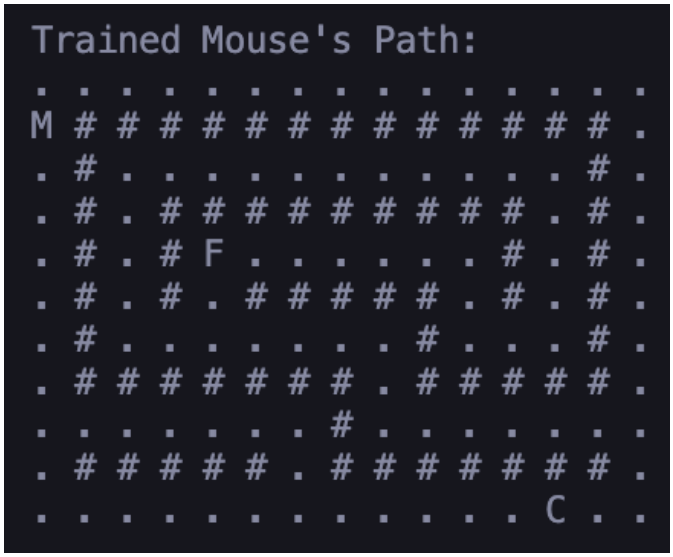
Analysis of Specific Episodes

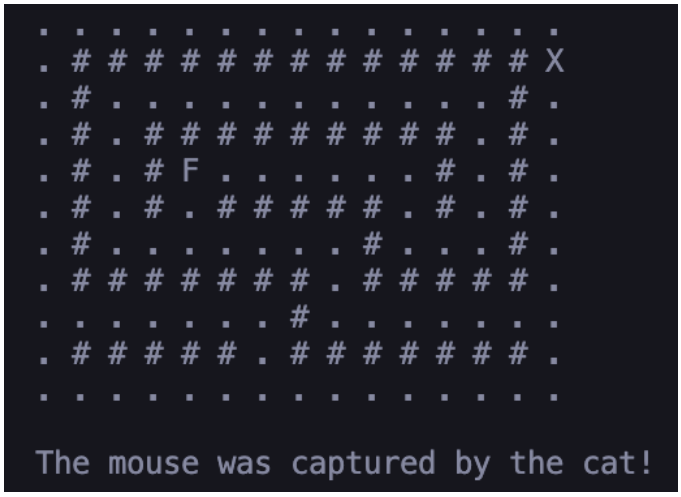
- **Early Episodes (1,000 - 5,000)**: Predominantly negative rewards suggest that the mouse is still learning to navigate the maze effectively. The higher ε values facilitate exploration, allowing the mouse to discover rewarding actions.
- **Mid to Late Episodes (6,000 - 10,000)**: Mixed rewards with occasional positive values indicate that the mouse has started to adopt more effective strategies. The low ε values imply that the mouse relies more on exploitation, utilizing the knowledge acquired during training.
- **Episodes 8,000 - 10,000**: The appearance of positive rewards alongside some negative ones demonstrates a balance between successfully reaching the food and facing challenges from the cat. This balance is crucial for maintaining a realistic and engaging simulation.

Example of Trained Mouse's Path

The trained mouse's behavior was evaluated in the enhanced maze environment. The following visualization illustrates the maze state during a test run, where:

- #: Wall.
- .: Free space.
- F: Food.
- M: Mouse.
- C: Cat.
- X: Mouse captured by the cat.





During the test run, the mouse navigates towards the food while the cat pursues. The balance between reaching the food and evasion is evident, as the mouse occasionally reaches the food (F) without being captured (X), while other times the cat successfully intercepts (C), preventing the mouse from achieving its objective.

Discussions and Conclusions

Analysis of Results

The implementation of Q-Learning in the Mouse and Cat game has proven effective in enabling the mouse to learn optimal navigation strategies within a complex maze environment. The convergence of the Q-table values indicates that the algorithm successfully identifies the most advantageous actions for the mouse to maximize its rewards. The enhancements made to the maze size and mouse movement logic have resulted in a more balanced and challenging scenario, where the mouse occasionally reaches the food while also being subject to capture by the cat.
The training results reflect a progressive improvement in the mouse's ability to navigate the maze. Initially, the mouse struggles to reach the food consistently, as evidenced by the predominantly negative rewards. However, as training progresses and the exploration rate decays, the mouse begins to exploit learned strategies, leading to occasional successful episodes where it reaches the food. The introduction of a larger maze and refined movement logic contributes to this balance, ensuring that neither the mouse nor the cat dominates consistently, thereby maintaining an engaging and realistic simulation.

Application of Theoretical Knowledge

This project effectively integrates core concepts of Reinforcement Learning, particularly Q-Learning. Key theoretical aspects such as state and action definitions, the ε-greedy policy for action selection, and the Q-table update mechanism have been meticulously applied. The project also addresses practical considerations, including the balance between exploration and exploitation and the careful design of the reward structure to guide the learning process of the mouse.

Distance Metrics: Manhattan Distance

The **Manhattan distance** plays a crucial role in the cat's movement strategy. Defined as the sum of the absolute differences of their Cartesian coordinates, it provides a straightforward metric for the cat to minimize the distance to the mouse in a grid-based maze. This distance metric ensures that the cat moves efficiently towards the mouse, enhancing its ability to capture or deter the mouse effectively.

$$\text{Manhattan Distance} = |x_{\text{cat}} - x_{\text{mouse}}| + |y_{\text{cat}} - y_{\text{mouse}}|$$

By leveraging the Manhattan distance, the cat adopts a strategic approach to pursue the mouse, making calculated moves that progressively reduce the distance between them. This deterministic strategy, combined with the probability of making an extra move, increases the cat's effectiveness in either capturing the mouse or blocking its path to the food.

Limitations and Future Improvements

- **Scalability**: The current state representation, which combines the positions of the mouse and cat, leads to a large state space that can become computationally expensive in larger mazes. Future work could explore dimensionality reduction techniques or more efficient state encoding methods to enhance scalability.
- **Adaptive Adversary**: Currently, the cat follows a deterministic strategy based on the Manhattan distance. Introducing adaptive behavior for the cat, such as learning-based strategies, could create a more dynamic and challenging environment, transforming the problem into a competitive multi-agent scenario.
- **Parameter Optimization**: The Q-Learning parameters (α, γ, ε) were selected empirically. Systematic optimization of these parameters, possibly through hyperparameter tuning techniques, could improve the learning efficiency and performance of the mouse.
- **Enhanced Reward Structure**: Refining the reward system to incorporate more nuanced incentives and penalties could lead to more sophisticated behavior patterns in the mouse, such as risk-averse strategies or long-term planning.
- **Visualization and Analysis Tools**: Developing more comprehensive visualization tools and analysis metrics could provide deeper insights into the learning process and behavior of both agents, facilitating better understanding and further enhancements.

Conclusion

The project successfully demonstrates the efficacy of Q-Learning in solving navigation and evasion problems within a complex environment. By accurately implementing the theoretical foundations of Reinforcement Learning and addressing practical challenges through maze expansion and movement logic improvements, the mouse agent effectively learns to balance the pursuit of its goal with the necessity of evasion. The resulting behavior showcases the potential of Q-Learning algorithms in developing intelligent agents capable of strategic decision-making in dynamic and adversarial settings.