Impacts of Reward Value Manipulation on the Behavior of a Reinforcement Learning Agent: A Case Study of a Gluttonous Snake Game

Author: Yijia Lu

Game code: https://github.com/Vinwcent/SnakeReinf.git

Abstract:

This report investigates the influence of altering reward values on the decision-making behavior of a reinforcement learning agent within a Python-based Gluttonous Snake game. Through meticulous manipulation and observation, the study identifies the nuanced changes in the agent's actions, revealing a complex relationship between reward configurations and gameplay outcomes. The results provide critical insights into the design of reward systems for reinforcement learning applications.

1. Introduction and Overview:

In this report, we explore this aspect of reinforcement learning through a practical case study involving a Gluttonous Snake game. A simple yet engaging environment, this game serves as an ideal platform for understanding the effects of different reward configurations on the agent's learning process.

2. Theoretical Background: Reward Systems in Reinforcement Learning

Reinforcement Learning (RL) is a machine learning approach where an agent learns to behave in an environment by performing actions and receiving feedback, known as rewards or penalties. These rewards guide the agent's learning process and shape its future behavior.

2.1 Reinforcement Learning Framework

In RL, the agent interacts with its environment in time steps. At each step, the agent receives a state s and selects an action based on its current policy. The environment transitions to a new state s' and provides a reward r to the agent. The agent's goal is to maximize the sum of rewards over time.

2.2 Role of Rewards

Positive Rewards: Encourage the agent to repeat beneficial actions.
Negative Rewards (or Penalties): Discourage the agent from repeating detrimental actions.
Reward Magnitude: Affects the strength of encouragement or discouragement.

2.3 Reward Shaping and Balancing

Reward shaping involves modifying the reward function to guide the agent towards the desired behavior. Careful balancing between rewards for different actions is crucial to avoid unintended consequences.

In this report, we investigate the impact of manipulating reward values in the Gluttonous Snake game, adjusting rewards associated with losing, inefficient, efficient, and winning moves to observe their effect on the game's outcome.

3. Algorithm Implementation and Development

The basic idea behind the test plan is to change one variable at a time, leaving the others constant. Then observe the effect on the snake's behavior. This is a rudimentary form of sensitivity analysis, a technique often used in complex systems to determine how different parts of the system respond to changes in parameter values.

Original dataset: `rewards = [100, 10, 50, 100000]`

```
# [Losing move, inefficient move, efficient move, winning
move]
```

We will change certain variables based on this original dataset.

Losing Move Reward: Vary this between high negative and zero values. [-100, 0, 200]. We want the snake to avoid losing moves, so the reward should always be negative or zero. High negative values should make the snake more cautious about colliding with itself or the game boundaries.

Inefficient Move Reward: We could set this to small negative, zero and small positive values [-10, 0, 200]. This will test how the snake responds to non-optimal routes to the apple. A small negative value should discourage inefficient moves, while a small positive value might cause the snake to make inefficient moves without much concern for the game's overall strategy.

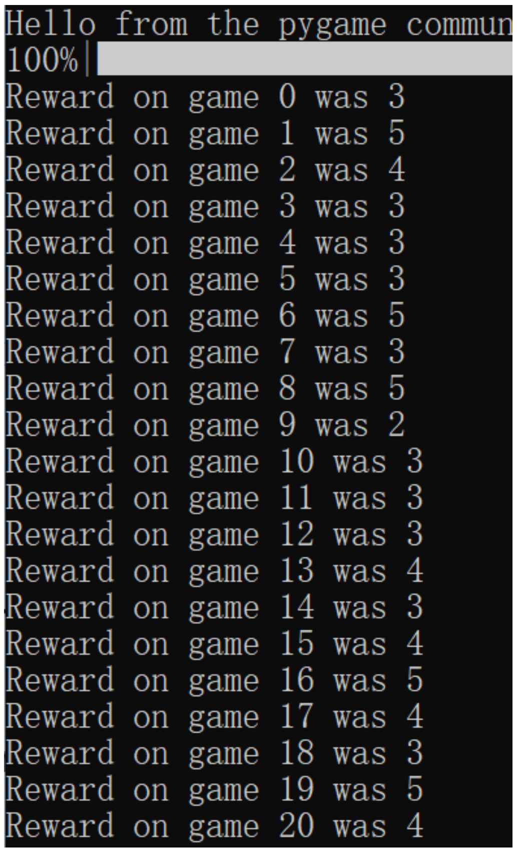
Efficient Move Reward: Vary this with positive values [0, 10, 1000]. The snake should always be rewarded for efficient moves. A higher value should make the snake more focused on directly reaching the apple.

Winning Move Reward: Test with high positive values [0, 100, 500000]. This is the reward for eating the apple and should be much higher than the other rewards to encourage the snake to prioritize eating the apple.

4. Computational Results

For our initial setting: `rewards = [100, 10, 50, 100000]`

We have the result output :



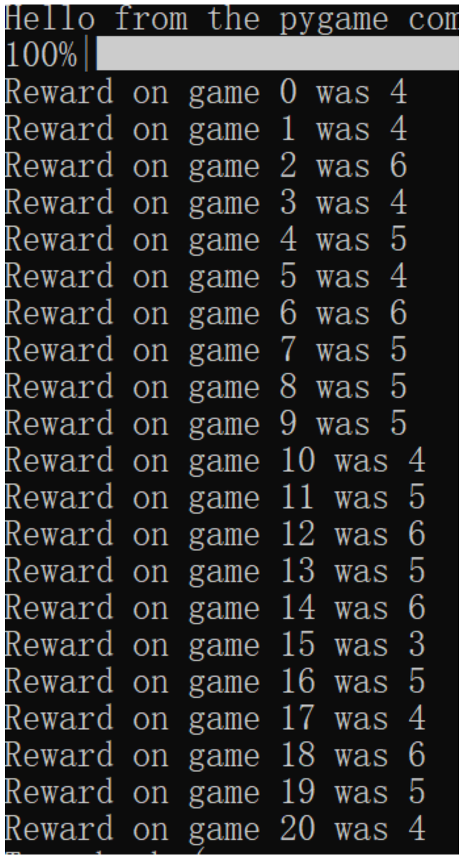
```
Hello from the pygame commun
100%|
Reward on game 0 was 3
Reward on game 1 was 5
Reward on game 2 was 4
Reward on game 3 was 3
Reward on game 4 was 3
Reward on game 5 was 3
Reward on game 6 was 5
Reward on game 7 was 3
Reward on game 8 was 5
Reward on game 9 was 2
Reward on game 10 was 3
Reward on game 11 was 3
Reward on game 12 was 3
Reward on game 13 was 4
Reward on game 14 was 3
Reward on game 15 was 4
Reward on game 16 was 5
Reward on game 17 was 4
Reward on game 18 was 3
Reward on game 19 was 5
Reward on game 20 was 4
```

We can calculate the average score under this setting is 3.85

1. Change Losing move: [-100, 0, 200]
    - Change Losing move to -100 and run the game

      In this situation, we found the snake just round around in the center and never ate an apple or tried moving close to the wall. The reason why it happened is because compared to winning moves, the loss move is too small and to win the game, the snake just tries not to do anything to survive.

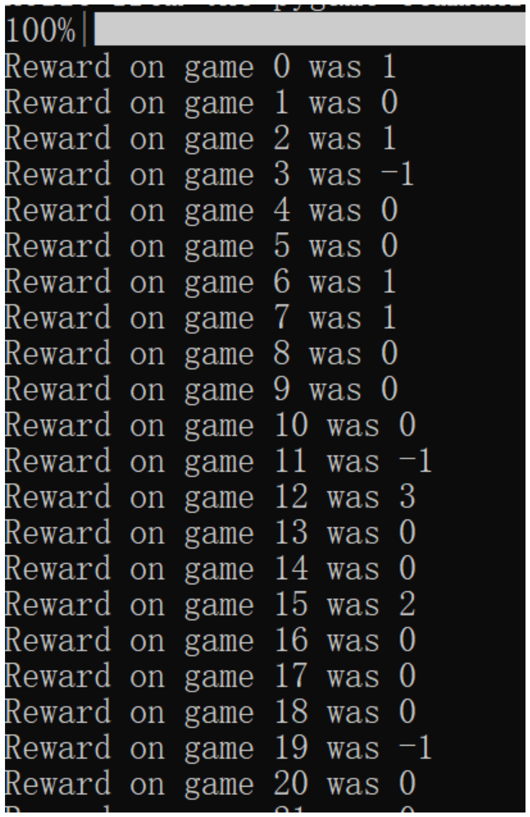
    - Change Losing move to 0 and run the game



```
Hello from the pygame com
100%|
Reward on game 0 was 4
Reward on game 1 was 4
Reward on game 2 was 6
Reward on game 3 was 4
Reward on game 4 was 5
Reward on game 5 was 4
Reward on game 6 was 6
Reward on game 7 was 5
Reward on game 8 was 5
Reward on game 9 was 5
Reward on game 10 was 4
Reward on game 11 was 5
Reward on game 12 was 6
Reward on game 13 was 5
Reward on game 14 was 6
Reward on game 15 was 3
Reward on game 16 was 5
Reward on game 17 was 4
Reward on game 18 was 6
Reward on game 19 was 5
Reward on game 20 was 4
```

We can calculate the average score under this setting is 5.05
With a zero losing move, our score before we die improves a lot.

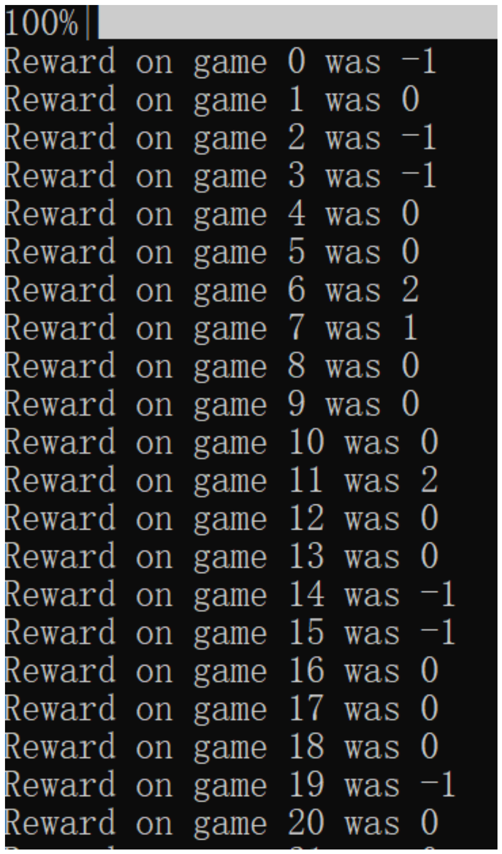
- Change Losing move to 200 and run the game



```
100%|
Reward on game 0 was 1
Reward on game 1 was 0
Reward on game 2 was 1
Reward on game 3 was -1
Reward on game 4 was 0
Reward on game 5 was 0
Reward on game 6 was 1
Reward on game 7 was 1
Reward on game 8 was 0
Reward on game 9 was 0
Reward on game 10 was 0
Reward on game 11 was -1
Reward on game 12 was 3
Reward on game 13 was 0
Reward on game 14 was 0
Reward on game 15 was 2
Reward on game 16 was 0
Reward on game 17 was 0
Reward on game 18 was 0
Reward on game 19 was -1
Reward on game 20 was 0
```

We can see that the score is pretty low. When the loss moves high, the snake will seeking for death from our observation.

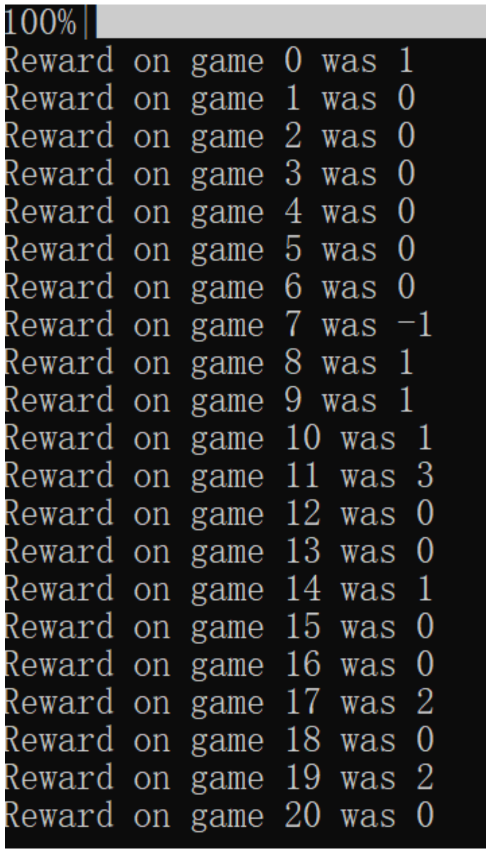
2. Change Inefficient move: [-10, 0, 200]
    - Change Inefficient move to -10 and run the game



```
100%|
Reward on game 0 was -1
Reward on game 1 was 0
Reward on game 2 was -1
Reward on game 3 was -1
Reward on game 4 was 0
Reward on game 5 was 0
Reward on game 6 was 2
Reward on game 7 was 1
Reward on game 8 was 0
Reward on game 9 was 0
Reward on game 10 was 0
Reward on game 11 was 2
Reward on game 12 was 0
Reward on game 13 was 0
Reward on game 14 was -1
Reward on game 15 was -1
Reward on game 16 was 0
Reward on game 17 was 0
Reward on game 18 was 0
Reward on game 19 was -1
Reward on game 20 was 0
```

When we run the game I can see the snake trying to eat the apple but always dies by touching the wall, maybe a low inefficient move because the snake doesn't learn how to find a good way to eat the apple. In this game both inefficient and efficient moves are important for snakes to learn.

- Change Inefficient move to 0 and run the game



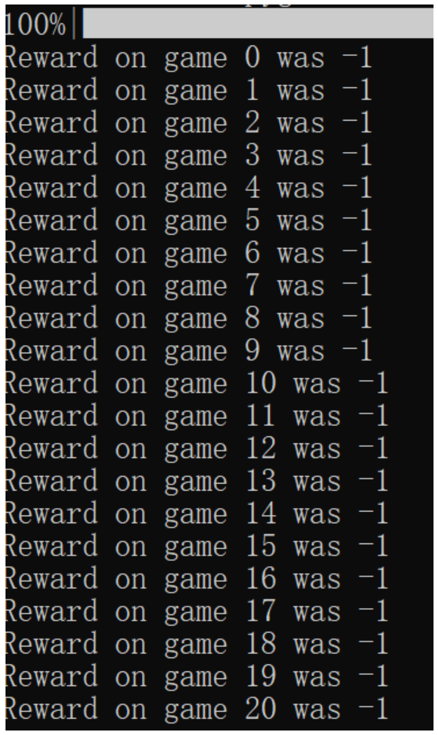
```
100%|
Reward on game 0 was 1
Reward on game 1 was 0
Reward on game 2 was 0
Reward on game 3 was 0
Reward on game 4 was 0
Reward on game 5 was 0
Reward on game 6 was 0
Reward on game 7 was -1
Reward on game 8 was 1
Reward on game 9 was 1
Reward on game 10 was 1
Reward on game 11 was 3
Reward on game 12 was 0
Reward on game 13 was 0
Reward on game 14 was 1
Reward on game 15 was 0
Reward on game 16 was 0
Reward on game 17 was 2
Reward on game 18 was 0
Reward on game 19 was 2
Reward on game 20 was 0
```

I think the reason for this display is similar to the previous one.

- Change Inefficient move to 200 and run the game

  In this situation, we found the snake just moved back forward in its initial place and never ate an apple or tried moving close to the wall. The reason why it happened is because, combined with high winning moves, the Inefficient move is also large so the snake just chooses to stay alive and not touch the apple.

3. Change Efficient move: [0, 10, 1000]
   - Change Efficient move to 0 and run the game



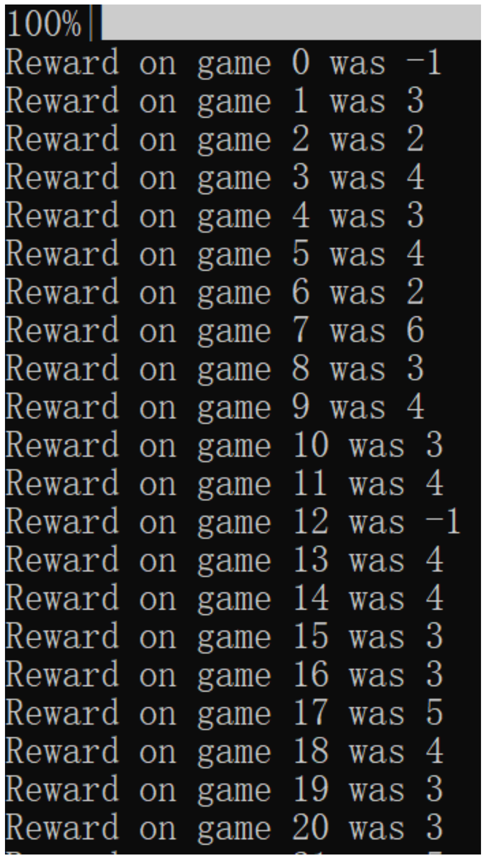
```
100%|
Reward on game 0 was -1
Reward on game 1 was -1
Reward on game 2 was -1
Reward on game 3 was -1
Reward on game 4 was -1
Reward on game 5 was -1
Reward on game 6 was -1
Reward on game 7 was -1
Reward on game 8 was -1
Reward on game 9 was -1
Reward on game 10 was -1
Reward on game 11 was -1
Reward on game 12 was -1
Reward on game 13 was -1
Reward on game 14 was -1
Reward on game 15 was -1
Reward on game 16 was -1
Reward on game 17 was -1
Reward on game 18 was -1
Reward on game 19 was -1
Reward on game 20 was -1
```

With 0 efficient move, the snake will not learn how to eat the apple and cause the above result.

- Change Efficient move to 10 and run the game

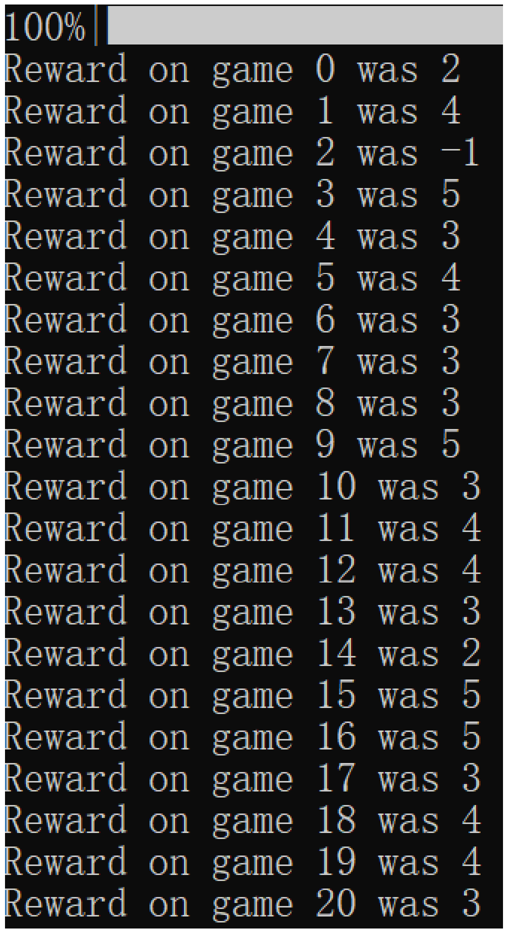
  The behavior of the snake in this situation is it moves back forward and will not try to find the way to eat an apple. This is probably because the reward for efficient and inefficient moves is the same, so the snake just can't figure out what to do inside the game.

- Change Efficient move to 1000 and run the game


```
100%|
Reward on game 0 was -1
Reward on game 1 was 3
Reward on game 2 was 2
Reward on game 3 was 4
Reward on game 4 was 3
Reward on game 5 was 4
Reward on game 6 was 2
Reward on game 7 was 6
Reward on game 8 was 3
Reward on game 9 was 4
Reward on game 10 was 3
Reward on game 11 was 4
Reward on game 12 was -1
Reward on game 13 was 4
Reward on game 14 was 4
Reward on game 15 was 3
Reward on game 16 was 3
Reward on game 17 was 5
Reward on game 18 was 4
Reward on game 19 was 3
Reward on game 20 was 3
```

The final result is similar to our initial setting, so we can find there is an upper limit about the efficient move training. Higher move reward can't reflect a better result output.

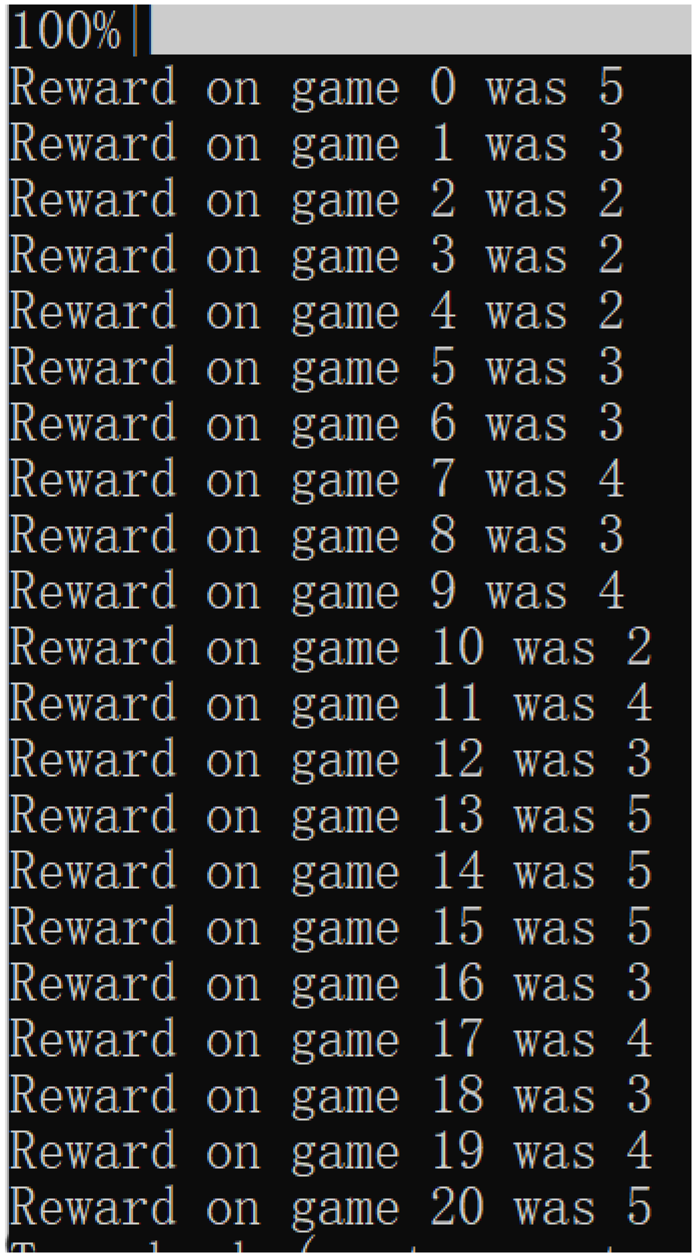
4. Change Winning move: [0, 100, 500000]
    - Change Winning move to 0 and run the game



```
100%|
Reward on game 0 was 2
Reward on game 1 was 4
Reward on game 2 was -1
Reward on game 3 was 5
Reward on game 4 was 3
Reward on game 5 was 4
Reward on game 6 was 3
Reward on game 7 was 3
Reward on game 8 was 3
Reward on game 9 was 5
Reward on game 10 was 3
Reward on game 11 was 4
Reward on game 12 was 4
Reward on game 13 was 3
Reward on game 14 was 2
Reward on game 15 was 5
Reward on game 16 was 5
Reward on game 17 was 3
Reward on game 18 was 4
Reward on game 19 was 4
Reward on game 20 was 3
```

That kind of surprise, with 0 winning move the final result is kind of the same. I think that with 0 winning moves, the snake learned to focus more on eating the apple. Also sometimes results like -1 appear because the snake didn't have the idea of avoiding touching the wall since we have 0 wins reward.

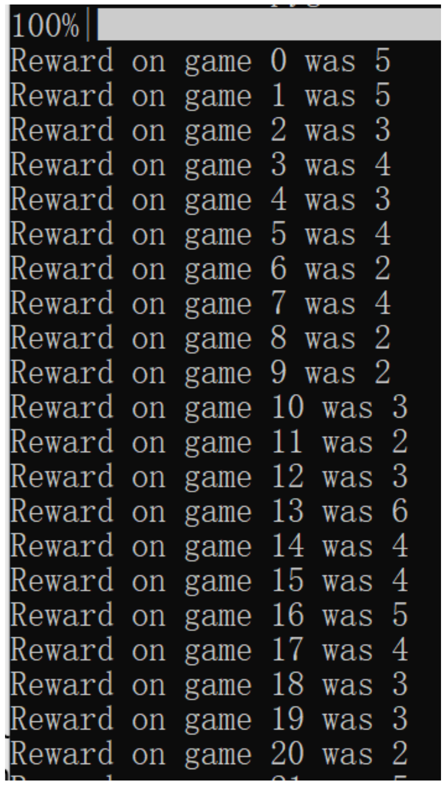
- Change Winning move to 100 and run the game



```
100%|
Reward on game 0 was 5
Reward on game 1 was 3
Reward on game 2 was 2
Reward on game 3 was 2
Reward on game 4 was 2
Reward on game 5 was 3
Reward on game 6 was 3
Reward on game 7 was 4
Reward on game 8 was 3
Reward on game 9 was 4
Reward on game 10 was 2
Reward on game 11 was 4
Reward on game 12 was 3
Reward on game 13 was 5
Reward on game 14 was 5
Reward on game 15 was 5
Reward on game 16 was 3
Reward on game 17 was 4
Reward on game 18 was 3
Reward on game 19 was 4
Reward on game 20 was 5
```

With a little bit higher winning move, the snake learns not to touch the wall and the situation of -1 score is displayed because of that.

- Change Winning move to 500000 and run the game



```
100%|
Reward on game 0 was 5
Reward on game 1 was 5
Reward on game 2 was 3
Reward on game 3 was 4
Reward on game 4 was 3
Reward on game 5 was 4
Reward on game 6 was 2
Reward on game 7 was 4
Reward on game 8 was 2
Reward on game 9 was 2
Reward on game 10 was 3
Reward on game 11 was 2
Reward on game 12 was 3
Reward on game 13 was 6
Reward on game 14 was 4
Reward on game 15 was 4
Reward on game 16 was 5
Reward on game 17 was 4
Reward on game 18 was 3
Reward on game 19 was 3
Reward on game 20 was 2
```

By increasing the winning move, the output result actually didn’t increase a lot, which shows that there is an upper limit of the reward of winning move.

5. The experiment's results illuminated the significance of each reward value:

Losing Move Reward: The value of this reward had a profound effect on the snake's survival strategy. When the penalty for losing was high, the snake tended to prioritize survival over eating the apple. In contrast, when the penalty was low, the snake played more aggressively, resulting in a higher average score.

Inefficient Move Reward: This reward significantly impacted the snake's path planning. With a lower value for inefficient moves, the snake often failed to find a safe and effective path to the apple. However, a higher value made the snake more cautious, avoiding risky moves and prioritizing survival.

Efficient Move Reward: This reward guided the snake towards the apple. A higher value did not necessarily lead to better performance, indicating a possible upper limit for its effectiveness. However, a zero value led to the snake not learning how to reach the apple, indicating its importance in the learning process.

Winning Move Reward: Changing this reward value had surprisingly limited impact on the snake's behavior. Even when the reward for winning was set to zero, the snake still learned to eat the apple. However, increasing this reward did not substantially improve the snake's performance, suggesting an upper limit for its effectiveness.

Role of Gamma in Learning

The experiments showed that the discount factor (gamma) played a crucial role in balancing exploration and exploitation. A lower gamma (0.6) appeared to encourage the snake to consider more immediate rewards and contributed to better gameplay with scores consistently ranging between 7 and 8. However, setting gamma too low (0.5) resulted in the snake occasionally ignoring the apple and just moving around, demonstrating the delicate balance needed for optimal gameplay.

5. Conclusion

The observations gathered suggest that the interaction between the reward parameters and gamma greatly influences the learning efficiency and gameplay quality in reinforcement learning. The ratio between different rewards and the choice of the discount factor (gamma) appear to be critical components for effective learning.

The chosen values of gamma and the rewards for the losing, inefficient, efficient, and winning moves appear to provide a robust setting for the snake to learn the game effectively. Increasing the number of episodes (n_episodes) could potentially lead to better gameplay as the snake agent gets more experience and refines its strategy.

In summary, the project underscores the importance of careful calibration of reward function and discount factor in reinforcement learning environments. The insights from this study could be beneficial in creating more effective reinforcement learning systems, which balance exploration and exploitation, risk and reward, and short-term and long-term gains.