
Snakes 3v3: VI or DQN?

Keyao You

Shanghai Jiao Tong University
youkeyao@sjtu.edu.cn

Yazhou Tang

Shanghai Jiao Tong University
tangyazhou518@sjtu.edu.cn

Abstract

In this project, we need to use our own agent to beat the opponent in the Snake 3v3 game on the Jidi AI platform¹. We have tried several algorithms to design our agent, like Minimax, Value Iteration, and Dueling DQN, while introducing specific strategies from the Snake game. Finally, we analyse the advantages and disadvantages of each algorithm, and choose the best-performed one.

1 Introduction

Snake 3v3 is a multiplayer game where players make decisions to control their snakes at the same time each turn (the number of snakes controlled by each player is 3).

1.1 Problem description

1.1.1 Game rules

According to information on the website, this game has several rules below.

1. There are 2 players in the game, each of them will control 3 snakes.
2. When snake's heads are in the body of its own or other snakes', this snake will be judged to be dead. It will then randomly reborn at any location on the map.
3. When snake eats a bean, its length will increase by 1. When a bean is eaten, it will randomly be generated anywhere on the map, thus ensuring that the total number of beans on the map is 5.
4. After 200 steps, the player with the larger sum of snakes' length wins.

1.1.2 Agent input and output

We need to write a Python program called `submission.py`. Here is the input provided by the game program and the output we need to prepare.

1. The first input observation is a dictionary with key from 1 to 7 and `board_width`, `board_height`, `last_direction`, `controlled_snake_index`. Here,
 - (a) 1 represents beans, while 2 to 7 represent snakes.
 - (b) The value of the dictionary is a list with `[h, w]` coordinates as elements, representing positions of beans or snakes. `h` represents the vertical distance from the origin in the upper left corner, `w` represents the horizontal width from the origin.
 - (c) Among the values corresponding to 2 to 7, the list elements from left to right represent the position of the head to the tail of the corresponding snake.

¹http://www.jidiai.cn/compete_detail?compete=13

- (d) The value of `board_width` is the width of the board; The value of `board_height` is the height of the board; The value of `last_direction` are directions of each snake in the last step; The value of `controlled_snake_index` is the controlled snake index.
2. The second input `action_space` is a list of available actions.
 3. The output is the actions that our snakes will perform in this turn.

1.2 Problem analysis

Our main goal is to avoid crash and let the snake eat as many beans as possible. So it is very intuitive to use adversarial search to solve this problem, or take it as a MDP problem and use value iteration to solve it. Besides, we need to mind that, we have 3 snakes instead of 1 snake, which means that it is not necessary for every controlled snake to pursue for longer length. So we have 2 kind of special cases.

- **DEFENSE.** When the snakes are too long, the possibility of crashing will increase. To avoid this kind of “sudden death”, maybe the snakes which achieve enough length don’t need to eat bean any more.
- **ATTACK.** When the snake dies in the second half of the race, it has a hard time catching up with its rivals in length again. Therefore it can move towards the opponent’s head, thus increasing the probability that the opponent will collide with it.

When 1 or 2 snakes behave specially, others will move normally, to ensure a steady rise overall.

2 Intuitive Algorithms

According to the analysis above, we tried several intuitive algorithms to solve the problem.

2.1 Minimax with α - β Pruning

2.1.1 Overview

Since this is a multi-agent problem, we can use minimax to search the game tree and decide the action if we suppose that all snakes take the action one by one so each layer represents a snake. According to the evaluation function, we simply design it as the difference between the sum of the lengths of snakes of each party. More details are shown in the code.

2.1.2 Result

The result of minimax agent with random agent as opponent is shown in figure 1.

```
Result base on 100 episodes:
+-----+-----+-----+
| Name   | minimax | random |
+-----+-----+-----+
| score  | 12.91   | 0.81   |
| win    | 88.0    | 4.0    |
+-----+-----+-----+
```

Figure 1: minimax agent with random agent

Here we set depth equal to two because it will cause a failure on Jidi platform if the depth is larger. Since minimax does not work well, we discard it and look for other methods.

2.2 Value Iteration

2.2.1 Overview

If we see each step as finding a way in a grid world, we can easily apply value iteration on it because the states are just the position in the grid world. In each step, we build a grid world in which the reward of bean and other snakes is defined by ourselves according to the observation. Then we use value iteration to count the Q value of each state and get the best policy. By the way, the reason why we do not use policy iteration is that it is a little bit slower than value iteration in this problem.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

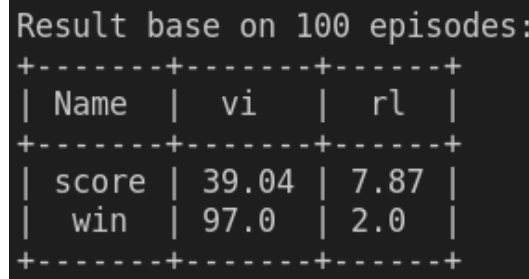
2.2.2 Special Strategies

We use some special strategies in the Snake game solve some tricky problems, enhancing our agent.

1. **Greedy problem.** The problem of this algorithm is that it is greedy to eat a bean and will easily get itself trapped. So we add a mechanism that it will follow its tail if it is long enough so that it will not die, which is a kind of defense.
2. **Collision problem.** Due to the disability to predict other snakes' action, it is likely to collide with other snakes' head. So we also give the surroundings of other snake' head a negative reward.
3. **Dead end problem.** This algorithm also has a problem of getting into a dead end if there is a bean in it. So we will change the bean reward to a negative value if three sides of it are snakes.
4. **Attack action.** If the score is too low, the snake will take the attack action which means they will rush to the surrounding of opponent snakes' heads. We just give the surroundings of opponent snake' head a positive reward.

2.2.3 Result

Its performance is shown in figure 2. Here we use the provided DDPG algorithm to train 50,000 episodes to get a model as our opponent, called "RL".



```
Result base on 100 episodes:
+-----+-----+-----+
| Name  | vi    | rl    |
+-----+-----+-----+
| score | 39.04 | 7.87  |
| win   | 97.0  | 2.0   |
+-----+-----+-----+
```

Figure 2: VI agent with RL agent

3 Dueling DQN

Although we used the VI algorithm to get a win rate of over 90% in the difficulty-boosting warm-up, we believe that we can use neural networks in combination with reinforcement learning to further improve the win rate.

3.1 Overview

If we see the problem as a classic MDP problem, then the state space will be too large which is difficult to store the Q value table. So we use deep Q network to learn the Q table and map

environment states to actions. To make the training more stable, there are two separate Q network. The target network is frozen during training and copied from evaluation network after several steps. What's more, all training samples are put into the replay buffer and training is performed on random samples from the replay buffer so that the relevance between samples is reduced. More details are shown in our code.

But the classic DQN has a problem of overestimating the Q value, so we use dueling DQN instead. It just splits the Q value into two parts, the state value and the advantage value. The state value reflects the value of the state and the advantage value reflects the value of each action. The advantage value is restricted to have an average value of 0 so that the Q value will be more appropriate.

3.2 Implementation

3.2.1 Input

The input is designed as a $3 \times 20 \times 20$ matrix. The first 20×20 matrix is the position of the controlled snake. The second 20×20 matrix is the position of other snakes. The third 20×20 matrix is the position of beans. The reason why the size is 20×20 instead of 10×20 is that the calculation of convolution performs better on 20×20 so we extend it. To show the possible moving of snakes, we also mark the surrounding of snakes' heads. To reduce the size of state space, we make the position of the controlled snake's head unchanged. The position of other snakes are calculated based on it.

3.2.2 Network design

We use two convolutional layers first since the input is image based. Then followed by a fully connected layer. The last layer is split to the state value function and the advantage value function. Finally combine them into the Q value. All activation functions use ReLU function. The network is shown in the figure 3.

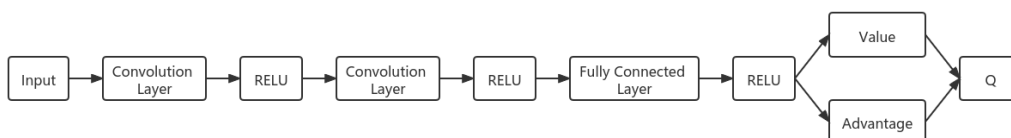


Figure 3: Network design

3.3 Training

We have tried many times to train the model. Here we have selected two representative attempts.

In one attempt, we first use random agent to train it. After it has learned enough, we change the opponent to RL agent. Finally we train it with itself to have a better performance. The reward and win rate during training are shown in figure 4 and figure 5.

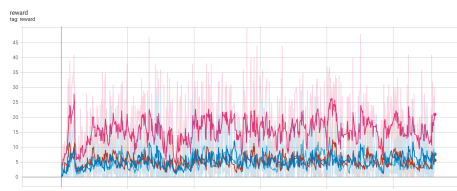


Figure 4: Reward during training

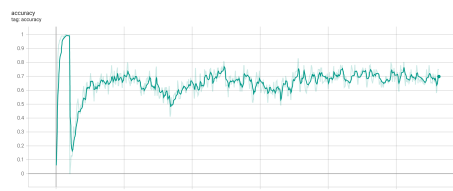


Figure 5: Win rate during training

In another attempt, we use VI agent to train it directly. The reward and win rate during training are shown in figure 6 and figure 7.

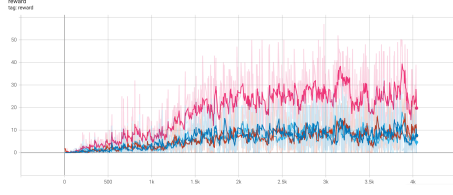


Figure 6: Reward during training

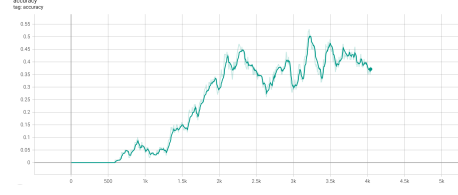


Figure 7: Win rate during training

3.4 Result

Its final performance is shown in figure 8. Here we use Value Iteration agent to be the opponent during training the Dueling DQN model. And we train for 4,000 episodes.

```

Result base on 100 episodes:
+-----+-----+-----+
| Name   | dqn   | rl    |
+-----+-----+-----+
| score  | 32.54 | 8.26  |
| win    | 99.0  | 1.0   |
+-----+-----+-----+

```

Figure 8: Dueling DQN agent with RL agent

4 Analysis

4.1 Advantages and disadvantages

Table 1: Advantages and disadvantages of 3 algorithms

Name	Implementation	Space needed	Performance
Minimax with α - β Pruning	Simple and intuitive	Large	Bad
Value Iteration & Policy Iteration	Simple and intuitive	Small	Decent
Dueling DQN	Relatively complex	Small	Maybe the best (If well-trained)

As shown in table 1, the 3 algorithms we have tried in this project have their advantages and disadvantages.

4.2 Comparison of VI and DQN

Although our DQN agent performed better against the RL agent, it was still the underdog against the intuitive VI agent. We think there may be several reasons for this.

1. The main reason is that we were not able to find better training parameters, resulting in unsatisfactory training results. As we know, tuning parameters is an important part of training models, which can directly affect the training rate and accuracy of the model. However, due to our lack of experience and the long time required for training, we were unable to find good training parameters.
2. At the same time, we use more special strategies in the VI agent. This is simple and straightforward, but also very effective. However, we cannot make the Dueling DQN agent acquire these strategies by directly adjusting the neural network structure, but we can only

hope that it will acquire them by itself. This process is full of uncertainty, and as a result, it fails to acquire these strategies.

4.3 Final agent

Due to the reasons above, we choose VI agent as our final agent. Its performance on the public channel² is shown in figure 9.

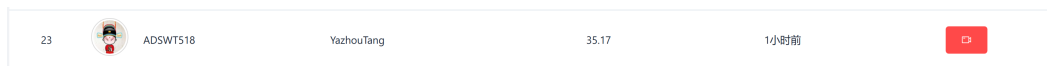


Figure 9: VI agent on Jidi platform

5 Summary

In completing this project, we both learned a lot while deepening our understanding of what we had learned in class. We tried several algorithms to solve the problem based on the analysis of the problem and from what we learned in class. The VI agent we designed on the first day already had good results, but we were not satisfied with that and continued to explore in two directions: on the one hand, we improved the performance of the VI agent by adding specific strategies, and on the other hand, we tried more complex Dueling DQN algorithms.

Although we were not able to find better training parameters for the DQN algorithm in the end, we learned a lot about deep learning and reinforcement learning in our continuous attempts, so these attempts are worthwhile.

In the end, we still chose the enhanced VI agent as the submission for the final competition because its intuitive algorithm was sufficient for most of the opponents.

Acknowledgments and Disclosure of Funding

We would like to thank Prof. Shuai Li and teaching assistants for their guidance and help throughout the semester. We have learned a lot in CS410 Artificial Intelligence course³ and have developed a strong interest in AI.

References

- [1] Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.
- [2] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
- [3] Zhou, Z. H. (2021). Machine learning. Springer.
- [4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
- [5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.
- [6] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016, June). Dueling network architectures for deep reinforcement learning. In International conference on machine learning (pp. 1995-2003). PMLR.

²http://www.jidi.ai/cn/ranking_list?tab=6

³<https://shuaili8.github.io/Teaching/CS410/index.html>

A Role and contribution of each member in the group

Keyao You: **60%**.

- Most of the code;
- Model training and testing;
- Report writing.

Yazhou Tang: **40%**.

- Least of the code;
- Model training and testing;
- Presentation;
- Report writing.

B Code

You can see the full code on <https://github.com/youkeyao/SJTU-CS410-Snakes-3V3-Group06>.