# CS410 Project: Snakes 3v3

**Haoyi You, Wenhao Chen**
Department of Computer Science and Engineering
Shanghai Jiao Tong University
{yuri-you, cccwher}@sjtu.edu.cn

## Abstract

Snake is a simple traditional game where players control the snake to eat beans and avoid death. As the number of snakes increases and the mode of the game changes into a competitive game, the difficulty of the game increases as well. Reinforcement Learning, however, has the ability to master many games. We applied the reinforcement learning into Snakes 3v3 and used various methods to train agents. In sum, we obtained some intelligent agents and achieved a good performance in the game. Code can be found at `https://github.com/CWHer/CS410-AI-project`.

## 1 Introduction

Snakes 3v3 is an improved version of the traditional snake game. Two-player alternate to control his 3 snakes and compete for a higher score. The game board of Snakes 3v3 has a shape of $10 \times 20$ with the boundaries that snakes can cross. Once a snake eats a bean, its length would increase one. Moreover, a snake is judged to be dead if its head collides with any snakes' bodies.

To train a more intelligent agent, we proposed several methods. Firstly, we tried to train the agent with Alpha Zero algorithm. Secondly, we applied DQN to overcome the drawbacks of Alpha Zero and carefully designed features and reward to make it compatible with a multi-agent environment. Then, we applied Imitation Learning to integrate the knowledge of different top agents. Finally, we combined our agents with some heuristic tricks to make them more stable and more robust.

## 2 Related Work

There has been a long history since people use Reinforcement Learning to manipulate games. In 2015, Mnih et al. [1] proposed Deep Q Network and used it to play the Atari game. And Deep Reinforcement Learning has been back in the spotlight ever since. Several improvements to DQN have been proposed later. In 2015, Ziyu Wang et al. [6] proposed Dueling DQN to make it more efficient and robust. And in 2016, Van Hasselt et al. [5] proposed Double DQN to solve the Q values overestimation issue.

Offline Reinforcement Learning methods for games have also been widely studied. In 2017, David Silver et al. [3] proposed Alpha Zero and achieved superhuman performance in chess-like games. Besides, Imitation Learning is also the focus of attention.

There are also some studies in the Snake 3v3 environment. Jiao Long Team [4] proposed a simple but powerful MLP DQN model. Jian Zhao et al. [7] proposed some carefully designed reward functions and used them to train a more advanced model. Moreover, Yuchen Shi [2] proposed some manually designed heuristic rules and used them to build an aggressive and robust agent.

# 3 Methodology

## 3.1 Offline RL: Alpha Zero

Considering that the environment is known and chess-like, we began our attempt with Alpha Zero [3].

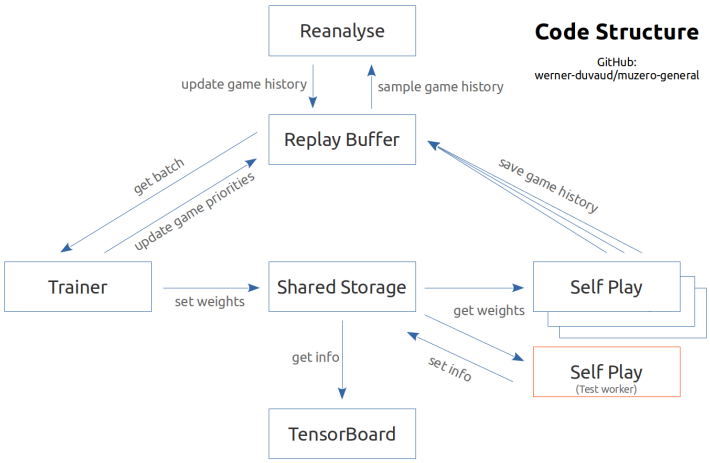


Figure 1: Code Structure

We managed to implement a toy Alpha Zero snake with a code structure similar to Fig. 1. However, in the end, we found that the training procedure is too slow and we had no more hardware to accelerate it. Still, it gives us some inspiration about code structure, network structure, and feature selection.

## 3.2 Online RL: Double Dueling Deep Q Network

After the first trial, we realized the shortcoming of Alpha Zero, so we were seeking a faster online RL approach. As both state space and action space are discrete, we decided to adopt Double Dueling DQN and we carefully designed features and reward to make it compatible with a multi-agent environment.

### 3.2.1 Algorithm Content

DQN [1] is the combination of deep neural network and Q-Learning, which intends to solve the representation and storage issues of exponential state-action space. Instead of directly storing $Q(s, a)$ table, DQN uses the neural network to fit Q value. Thus, DQN consumes less memory and has a good generalization ability.

The Q value update rule of DQN is similar to Q-Learning.

$$Q_\omega(s, a) \leftarrow r + \gamma \max_{a' \in \mathcal{A}} Q_\omega\left(s', a'\right)$$

However, Q values are often overestimated in DQN. In order to solve this issue, Double DQN algorithm [5] was proposed. Double DQN separates train network and target network and uses a new formula to update Q value.

$$Q_\omega(s, a) \leftarrow r + \gamma Q_{\omega^-}\left(s', \underset{a'}{\operatorname{argmax}} Q_\omega\left(s', a'\right)\right)$$

To make DQN more efficient and stable, Dueling DQN algorithm [6] was proposed. Dueling DQN separates Q value function into value function and advantage function. Therefore, the value function can be updated more frequently.

$$Q_{\eta,\alpha,\beta}(s,a) = V_{\eta,\alpha}(s) + A_{\eta,\beta}(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A_{\eta,\beta}(s,a')$$

We integrate both double and dueling techniques into our implementation of DQN and use soft-update to update the target network more smoothly.

### 3.2.2 Implementation Details

#### Action Design

As it is meaningless to go in the opposite direction to the current one, we mask this invalid action to reduce the problem scale.

#### Feature Design

Instead of manually extracting some linear features, we leverage the original observation and separate it into 11 channels. In addition, we add 1 channel to record the progress of the game, since the entire game is controlled by the number of steps. Tab. 1 shows contents of different channels.

Table 1: Features

| Channel | Content |
|---|---|
| $1-3$ | historical positions of opponent in last 3 steps |
| $4-6$ | historical positions of team in last 3 steps |
| 7 | position of controlled snake's head |
| 8 | positions of controlled snake |
| 9 | positions of other teammates' heads |
| 10 | positions of opponents' heads |
| 11 | positions of all beans |
| 12 | progress of the game |

However, as the direction of the controlled snake head is varied, it is quite hard to learn the mapping between features and all valid actions. We managed to solve this issue with **rotation trick**. First, we move the head of the controlled snake to the center of the map. Then, we rotate the head direction, which makes it always toward the up direction. Finally, as the map is anisotropic, we have to clip features into $10 \times 10$ square to achieve a uniform feature shape. Features encoding can be better illustrated by Fig. 2 and Fig. 3

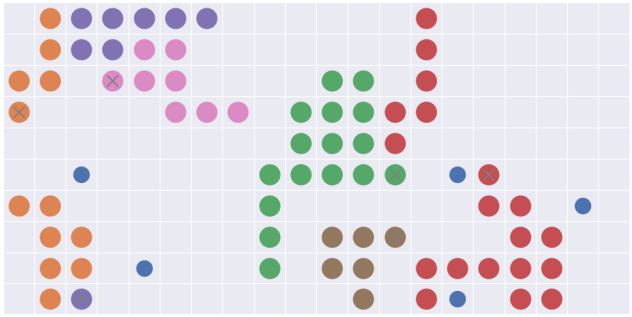

Figure 2: Sample State with Step = 100

#### Network Structure

Considering that the input features are 2-dimension, we apply convolution neural network to deal with them. Moreover, we apply residual network block, as it can void deep neural network degradation. The network structure is illustrated in Fig. 4.

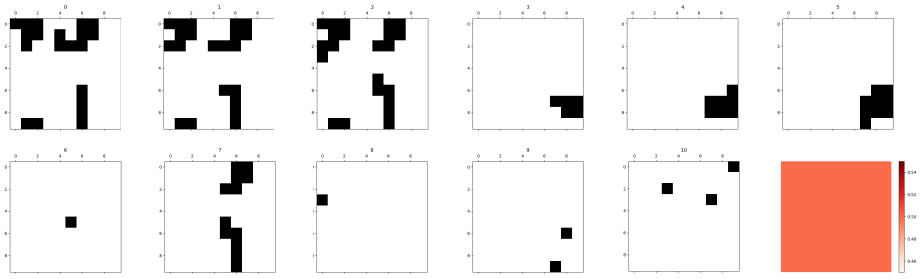

Figure 3: Encoded Features of Yellow Snake

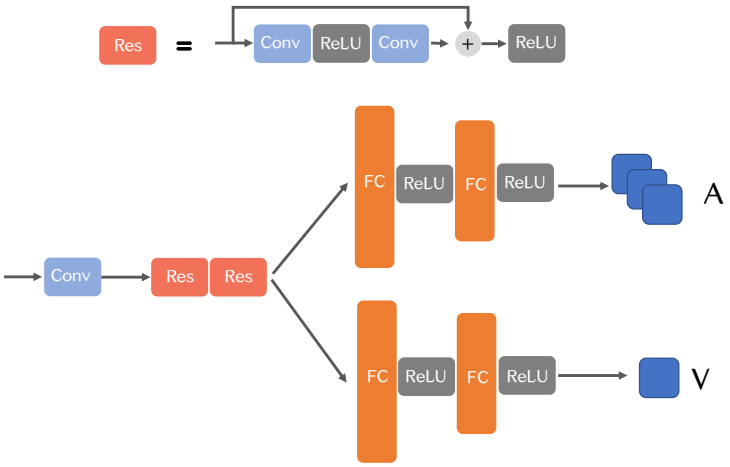


Figure 4: Network Structure

**Reward Design**

Since the total length of a single episode is 200, we set $\gamma = 0.98$ to make the agent foreseen more.

We refer to the well-designed reward of Jian Zhao et al. [7], which first formulates a zero-sum member reward $r_i$ and uses it to further calculate a team-related step reward $r_i^\star$. Notations are shown in Tab. 2 and the step reward is calculated according to the following formula.

Table 2: Notations Used to Formulate Reward

| Notation | Meaning |
|---|---|
| $\Delta r_i$ | reward of snake $i$ directly given by the environment |
| $\bar{\Delta r}'$ | average $\Delta r_i$ of opponent |
| $\bar{r}$ | average $r_i$ of team (i.e. team reward term) |
| $\alpha$ | weight of team reward term |

$$r_i = \Delta r_i - \bar{\Delta r}'$$
$$r_i^\star = (1-\alpha)\cdot r_i + \alpha \cdot \bar{r}$$

Besides, we set the episode reward to $\pm 20$ to make it weighs more much than the step reward.

**Training Procedure**

As the environment needs more than one player, we apply self-play and evolution in the training procedure. The training procedure is illustrated as follows.

1. Use self-play to generate training data and store them in replay buffer.
2. Sample data from replay buffer and use them to train the current model.
3. After repeating step 1 and step 2 several times, evaluate the current model against the historical best model. Update historical best model if current model has a better performance.

### 3.3 Offline RL: Imitation Learning

Double Dueling Deep Q Network is able to obtain a relatively good performance, but it is still some way from state-of-the-art models. We believe that the limited feature shape causes degradation of DQN. However, there is a trade-off between features shape and training efficiency. We apply Behavioral Cloning, one of the Imitation Learning methods, to solve this issue.

#### 3.3.1 Algorithm Content

The idea of Behavioral Cloning is relatively simple. First, separate all the trajectory data into state-action pairs. Then, train a model to predict the action given state. Since the existence of compounding error, it is tricky to train a model with good performance.

#### 3.3.2 Implementation Details

**Action Design**

Considering that the prediction problem is much easier than directly learning from the environment, we do not mask any action. Thus, the model can still manipulate the mapping between features and actions without **rotation trick**.

**Feature Design**

We use the same feature in Table. 3, but without **rotation trick**.

**Network Structure**

We deepen the model since the dimension of features is larger. Besides, we apply circular padding in convolution neural network as snakes can cross boundaries. The network structure is illustrated in Fig. 5.

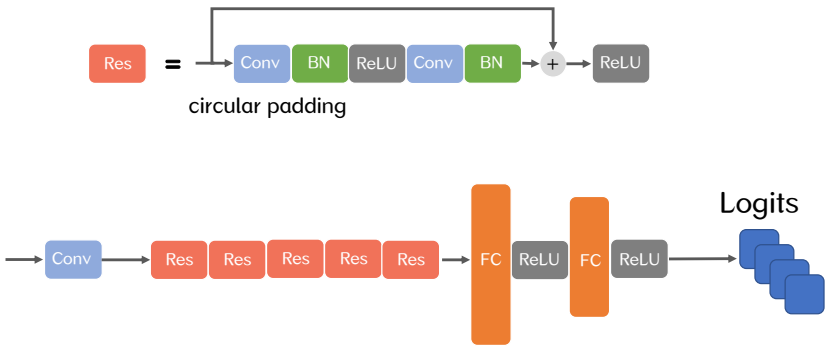


Figure 5: Network Structure

**Training Procedure**

The training procedure is a little tricky and it can be divided into two parts.

1. Use DQN to generate data and train the imitator. Repeat this process several times until the imitator can predict with 90% precision.
2. Fine-tuning the imitator with filtered top players' game data.

### 3.4 Heuristic Tricks: Negative Collision Avoidance and Active Defense

After evaluating agents, we discovered some common problems of collision. Thus, we decided to combine agents with several heuristic methods to improve the performance.

#### 3.4.1 Negative Collision Avoidance

In accordance with the game rule, there are two types of collision. One is that the snake head collides with any snake bodies and the other is that the snake head collides with another snake's head. The former would only cause the death of the controlled snake while the latter will cause the death of both colliding snakes. Both types of collision degrade team score, except when the length of the controlled snake is less than the opponent that collides with. Therefore, when choosing an action for the controlled snake, we need to avoid the so-called **negative collision**, which can be divided into the following 3 cases.

1. Controlled snake collides with any snake bodies.
2. Controlled snake collides with teammates.
3. Controlled snake collides with an opponent that has less length.

We use mask to avoid these dangerous actions and the procedures are as follows.

1. Mask all the snake bodies, including heads but excluding tails.
2. Mask teammates move. Information of teammates is passed through global variables.
3. Mask potential attacks of opponents that have less length.

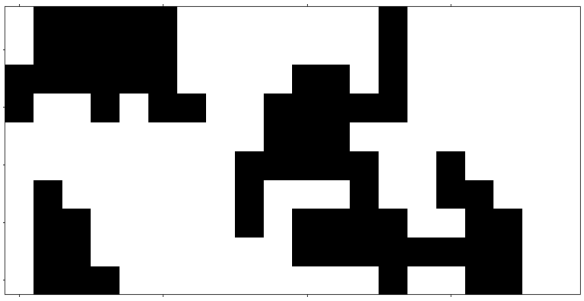

Figure 6: Mask of Yellow Snake

The Mask procedure can be better illustrated by Fig. 2 and Fig. 6.

#### 3.4.2 Active Defense

Negative collision avoidance is able to improve some performance of agents. However, there are times when agents are too greedy and put themselves into a dilemma. As is illustrated in Fig. 7, agents might sometimes choose to go the right direction and finally block and kill themselves.

Moreover, as the length of snakes increases, the cost and probability of death also increase but the benefit of eating beans just keeps the same. Therefore, it is better for the longer snake to perform some defense actions rather than seeking beans. The RL agents have little awareness of defense, so we implemented an active defense module and insert it into RL agents. Yuchen Shi's work [2] gives us some insights, we followed his algorithm framework and made some improvements.

**Preliminary Definition**

**Safe Position** Position $s$ is safe for snake $i$ if and only if snake $i$ can reach position $s$ without collision and earlier than any other snakes with less length.

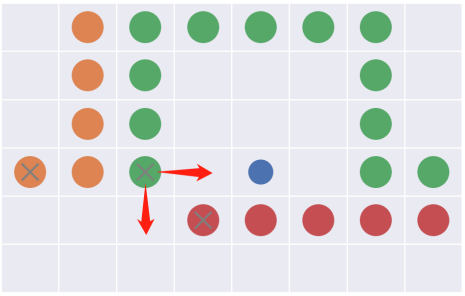

Figure 7: Greedy Dilemma

**Algorithm Content**

**Multi-Source BFS** We apply multi-source BFS to compute safe position for each snake. Multi-source BFS takes snake heads as sources and snake bodies as obstacles. Besides, snakes with less length take priority over longer ones. Note that when the BFS is in progress, the tail positions of snakes would become empty after each round and can be occupied by other snakes, since snakes are stepping forward.

**Tail Biting with Interval** Consider such a situation where the snake's head is adjacent to its tail and the snake always moves head towards the tail. We refer to this situation as Tail Biting. In such a situation, the death probability of a snake is relatively low. The only way to kill such a snake is to precisely snipe its head with another snake's head. However, it is almost not possible to plan this. Even though the sniping is successful, it brings loss to the opponent as well.

Thus, the main object of Active Defense is to bring a snake into Tail Biting state. As a matter of fact, if the snake can move to the last $i^{th}$ position within $i$ steps, it has a great chance to successfully enter Tail Biting state. To avoid collision in the halfway, we must also ensure that position is a safe position of the current snake.

Unfortunately, the snakes are unlikely to enter the ideal Tail Biting state as the conditions are too strict. However, we noticed that if there are some intervals between head and tail, the snake is defensive and as long as its head moves towards its tail. Thus, we improved Tail Biting by weakening conditions and we call this Tail Biting with Interval. The interval $t$ is a discrete hyper-parameter. And we bring a snake into Tail Biting with Interval state if the snake can move safely to the last $i^{th}$ position within $i + t$ steps.

## 4 Experiment and Result

Since the training procedure is quite time-consuming, we did not have enough time and hardware to conduct ablation study of every hyper-parameter. Instead, we only chose several hyper-parameters that we believe are of great importance.

### 4.1 DQN Hyper-parameters

**Impact of Feature Rotation Trick**

Feature rotation trick simplifies the mapping between features and valid actions.

As is illustrated in Fig. 8, rotation trick can largely reduce training time while maintaining a good performance.

**Impact of Team Reward Weight $\alpha$**

Team reward weight $\alpha$ controls the level of the agent focusing on its teammates.

As is illustrated in Fig. 9, smaller $\alpha$ slightly accelerates the training process but results in a slightly inferior performance.

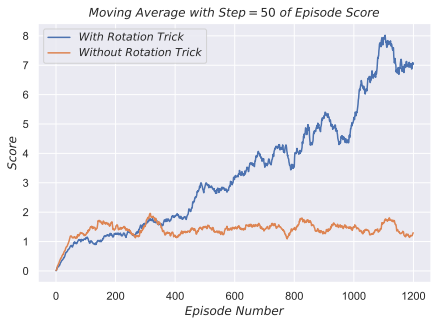


Figure 8: Impact of Rotation Trick


Figure 9: Impact of $\alpha$

### 4.2 Imitation Learning Hyper-parameters

**Impact of Fine-tuning Data Quality**

In the fine-tuning stage of the imitator, we filter top players' data according to the pre-set reward threshold. In other words, we only choose trajectories that can result in good rewards.

Table 3: Impact of Reward Threshold.

| Reward Threshold | 35 | 40 | 45 | 50 |
|---|---|---|---|---|
| Win Rate against Greedy (both without Active Defense module) | 43% | 53% | 49% | 45% |

As is illustrated in Tab. 3, either too low or too high thresholds lead to performance degradation. Since both underperform episodes and high-quality episodes between top players can result in fewer rewards, there is a trade-off between them.

## 5 Conclusion

In summary, we tried several reinforcement learning algorithms and heuristic tricks in Snake 3v3 game. Though there were some trials and errors, we overcame them by applying new models and new tricks. Finally, we were able to obtain relatively good performance and made our agent intelligent and robust.

Without doubt, the methods proposed in our paper still have room for improvement. More complicated multi-agent RL methods, such as QMIX and MAPPO, are likely to achieve a better performance than DQN. Besides, more ablation studies with different initial seeds can be conducted to determine whether some modules are necessary and tune the model accordingly.

## 6 Group Member Contribution

- Haoyi You: Implement Active Defense. Write the corresponding part and the preliminary parts of the final report.
- Wenhao Chen: Implement Alpha Zero, DQN, Imitation Learning, and Negative Collision Avoidance. Write the corresponding parts and the experiment part of the final report.

## 7 Reference

## References


[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Yuchen Shi. Heuristic snakes algorithm. `https://yuchen.xyz/2021/08/21/Jidi%20Snakes%203V3/#more`, 8 2021.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[4] Jiao Long Team. Jiao long snakes algorithm. `https://gitee.com/jidiai/rlchina2021/attach_files`, 8 2021.

[5] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[6] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[7] Jian Zhao, Dongyun Xue, and Zihan Li. Hou lang snakes algorithm. `https://gitee.com/jidiai/rlchina2021/attach_files`, 8 2021.