Mastering the Game of 3v3 Snakes with Rule-Enhanced Multi-Agent Reinforcement Learning

1st Jitao Wang
University of Science and Technology of China
Hefei, China
wangjitao@mail.ustc.edu.cn

2nd Dongyun Xue
University of Science and Technology of China
Hefei, China
andyxue@mail.ustc.edu.cn

3rd Jian Zhao
University of Science and Technology of China
Hefei, China
zj140@mail.ustc.edu.cn

4th Wengang Zhou
Institute of Artificial Intelligence
Hefei Comprehensive Nation Science Center;
University of Science and Technology of China
Hefei, China
zhwg@ustc.edu.cn

5th Houqiang Li
Institute of Artificial Intelligence
Hefei Comprehensive Nation Science Center;
University of Science and Technology of China
Hefei, China
lihq@ustc.edu.cn

***Abstract*—As a popular game around the world, Snakes has multiple modes with different settings. In this work, we are dedicated to the 3v3 Snakes, which is characterized by a complex mixture of competition and cooperation. To address this mode of Snakes, most existing AI agents adopt rule based methods, which achieve limited performance due to human's oversight of some special circumstances. Inspired by the superiority of multi-agent reinforcement learning (MARL), we propose a rule-enhanced multi-agent reinforcement learning algorithm and build a 3v3 Snakes AI. Specifically, we introduce the territory matrix which is commonly utilized in rule based methods to the state features and mask the illegal actions through designed rules. The relationships of individual-team and friends-foes are also merged into reward design. Trained with Distributed PPO and self-play on a single GeForce RTX 2080 GPU for twenty-four hours, our AI achieves state-of-the-art performance and beats human players. On JIDI platform, our agent outperforms the other 132 participating agents and ranks the first for more than 20 consecutive days.**

***Index Terms*—Snakes, Multi-Agent System, Game, Reinforcement Learning**

I. Introduction

Games are often abstracted from real-world problems, which makes them natural benchmarks for intelligent decision algorithms. With the development of reinforcement learning (RL), recent years have witnessed significant achievements in various single-player games such as Atari [1] and Super Mario [2]. Meanwhile, multi-agent games have caught more and more attention from researchers, where the agents may compete or cooperate with others. Specifically, encouraging advance has been made in competitive games. For example, the performance of AI in Mahjong [3], multi-player Texas Hold'em [4] and Fighting game [5] have reached human professional level. As for cooperative multi-player games, StarCraft Multi-Agent Challenge (SMAC) [6] and Overcooked [7] have become important testbeds for research of multi-agent reinforcement learning. Apart from the above games, current research efforts are turning to more challenging multi-player games which simultaneously include cooperation and competition. OpenAI, DeepMind and Tencent have proposed their game AI in DOTA [8], StarCraft [9] and Honor of Kings [10] and achieved amazing achievements, which arouses widespread attention.

This work is dedicated to building an AI program for 3v3 Snakes. Snakes is a popular game across the world, where teams manage to make the snakes longer by controlling the moving direction of the snakes and letting snakes eat beans. In the basic 1v1 Snakes game, once the head of a snake hits itself or the body of any other snake, it will be immediately killed and then reset to a new life with an initial short length somewhere in the map.

3v3 Snakes not only keeps the basic elements of Snakes, but also introduces competition and cooperation in the game. Each team here controls 3 snakes at the same time. In the game, every snake should not only eat beans, but also cooperate to defend themselves and discourage other snakes from growing longer. When the maximum episode step is reached, the snake with longest length becomes the winner. Compared to other multi-agent environments like SMAC [6], another advantage of 3v3 Snakes is that the simulation of the game requires little computation. Every step of the game only takes time in the order of millisecond and the simulation can be done even on personal computers with considerable speed, making it an

This work was supported in part by the National Natural Science Foundation of China under Contract 61836011 and in part by the Youth Innovation Promotion Association CAS under Grant 2018497.

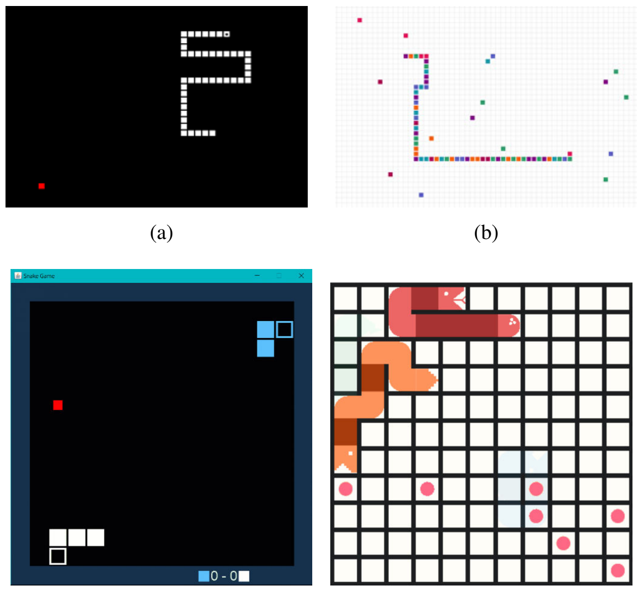


Fig. 1: Several Snakes game modes. (a) is a game mode with only one snake and one bean; (b) is a game mode with one snake and multiple beans; (c) is a game mode with multiple snakes and one bean; (d) is a game mode with multiple snakes and multiple beans.

ideal environment to evaluate MARL algorithms.

In the traditional rule-based 3v3 Snakes program, a concept of a territory matrix is introduced. Specifically, the map of the game is first split into several pieces, each of which representing the territory of a snake. Based on these territories, the snakes try to eat beans by heuristic commands. If any snake grows to a certain length, the algorithm will direct it to form a circle using its body (its head adjacent to its tail) to defend itself. With the development of MARL, there have been some 3v3 Snakes algorithms based on MARL methods. However, these algorithms fail to achieve a comparable performance against rule-based algorithms. In this work, we take advantage of both rule and MARL and proposed a novel 3v3 Snakes AI. We make our contributions from two aspects. First, we build benchmarks for 3v3 Snakes to help further investigation. Second, we propose a rule-enhanced MARL algorithm. To better integrate environment information, we add the territory matrix, the core part of rule-based algorithms, into feature matrices. We also devise a zero-sum individual reward as well as a team reward to encourage cooperation among snakes in the same team. To train the agents, we apply the distributed proximal policy optimization (PPO) paradigm [11]. After training on a single GeForce RTX 2080 GPU for 24 hours, our agent is able to achieve state-of-the-art performance and beats human players. We test our algorithm on the popular JIDI platform[1] against 132 competitive participating agents, and has kept the first place for more than 20 consecutive days among all these competitors.

II. Related Work

In this section, we briefly introduce the concept and formulation of reinforcement learning, and review its application in games.

A. *Reinforcement Learning*

Reinforcement learning (RL) [12] intends to find the agent's optimal policy from the interaction between the agent and the environment. Generally researchers use the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ Markov Decision Process (MDP) to model the learning process, which represent the state space, action space, reward function, transition probability function and discount factor, respectively. In the interaction, at each time step t, the agent observes the environment state $s_t \in \mathcal{S}$, and then selects an action $a_t \in \mathcal{A}$ according to the current policy $\pi(a_t|s_t)$. After that, the environment would transfer to the next state s_{t+1} according to the state transition probability function $P(s_{t+1}|s_t, a_t)$ and the agent receives a reward $r_t \in \mathcal{R}$. The whole process is defined as a trajectory τ, and the cumulative reward of this trajectory is as follows:

$$R_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k. \tag{1}$$

During the optimization process, an RL agent is aimed to find the optimal policy to maximize the cumulative reward expectation $J(\pi)$:

$$J(\pi) = \mathbb{E}_{s\sim\mathcal{P}, a\sim\pi}[\sum_{t=0}^{T} \gamma^t r_t]. \tag{2}$$

In order to find the optimal policy, lots of DRL algorithms have been proposed, such as Trust Region Policy Optimization (TRPO) [13], Proximal Policy Optimization (PPO) [14],Deep Deterministic Policy Optimization (DDPG) [15], Rainbow [16] and so on.

B. *Reinforcement Learning in Games*

Conventional rule-based game AI approaches have difficulty in handling complex state information such as images. In order to overcome this problem, witnessing the fast development of DRL in recent years, many researchers have adopted the DRL methods to build the game AI. In the seminal work by Mnih et al [1] , researchers combine deep neural networks with the Q-learning [17] algorithm training Deep Q-Networks (DQNs) with Experience Replay Buffers to obtain human-level performance on multiple classic Atari 2600 video games. In [18], DeepMind uses the Monte Carlo Tree Search (MCTS) to enable DRL agent to search optimal strategy, and defeats the top human chess player Le Sedol four to one in Go, which

[1]JIDI is a reinforcement learning evaluation platform which is developed by the Institute of Automation, Chinese Academy of Sciences: http://www.jidiai.cn/ranking_list?tab=6.

was once considered impossible to be conquered by AI due to its extremely complex state space. In a famous Real Time Strategy game StarCraft, DeepMind designs the RL game AI AlphaStar [9] to beat many top human players. Besides, some works have bring RL into multiple players games. Recently, the OpenAI and Tencent have used the reinforcement learning methods to build their game AI in different Multiplayer Online Battle Arena (MOBA) games, *i.e.*, Dota [8] and Honor of Kings [10], achieving impressive performance.

III. Preliminary

A. 3v3 Snakes

Snakes, a video game published in 1976, has been popular around the world for decades. In the classic mode of the game, the human player has to control the move of a snake in a grid map to eat beans. The snake will grow longer by one unit every time it eats one bean. The goal of this game is to control the snake to grow as long as possible without hitting its body. There have been some works solving this task by utilizing reinforcement learning techniques [19]–[21]. Our work mainly aims to solve a complicated version of this game, *i.e.*, 3v3 Snakes. It involves both cooperation and competition simultaneously, thus raising the difficulty of this game and making it challenging for the development of game AI.

At the beginning (time step 0), 6 three-unit-long snakes emerge at random locations in a grid map with a height of 10 and a width of 20. The snake can move towards four directions (up, down, left, right) and becomes one unit longer when it eats a bean. However, when a snake bumps into itself or another snake, it will regenerate in the map with a length of 3 units. What's more, there are always 5 beans on the map and if one bean is eaten, a new bean will appear anywhere on the map where there are no snakes. After 200 time steps, the system will calculate the sum of the three snakes in each side and the one with longer length wins.

B. Territory Matrix

The current best-performing rule-based method requires calculating a territory matrix. In this section, we introduce how the territory matrix is calculated and the detailed procedure is shown as follow:

- Get the coordinate of each snake in the 10×20 grid map, including the positions of their bodies and heads.
- Delete the tail of each snake and mark the grid that is adjacent to the head of one snake as (index of the snake [0-5], 1). Notable, such grid should not be occupied by bodies of any snake. If any grid is occupied by two or more snakes at the same time, mark it as (- number of conflicting snakes +1, 1).
- Delete the tail of each snake and mark the grid that is adjacent to grid with label 1 as (index of the corresponding snake [0-5], 2). Similarly, such grid should not be occupied by bodies of any snake and has no label initially. If any grid is occupied by two or more snakes at the same time, mark it as (- number of conflicting snakes +1, 2).
- Repeat above operations until all grids on the map are marked. At this time, an attack range matrix is acquired as shown in Figure 2.

After getting the attack range matrix, we need to encode it into a territory matrix. From above description, we can know that the first element of the label of each grid occupied by more than one snake is negative. We call the value of each grid, whose label in attack range matrix is (i, j), in territory matrix N and the encoding process is shown in Equation 3:

$$N = \begin{cases} 6 * j + i, & i \in [0, 5] \\ -6 * j + i, & i < 0 \end{cases} . \tag{3}$$

In this way, we can obtain the territory matrix, which is shown in Figure 3(b).

IV. Method

In this section, we introduce the structure of our AI agent, including input feature, network structures, reward design and training paradigm.

A. Features design

The map in 3v3 Snakes is fixed to be a 10×20 matrix. Based on this, we derive a $12 \times 10 \times 20$ matrix from the environment as the input feature. The first 11 channels are one-hot feature with physical meanings listed as follows:

- $[1]$: the full body positions of the six snakes.
- $[2]$: the head position of the currently controlled snake.
- $[3]$: the head positions of the three snakes in our team.
- $[4]$: the head positions of the three snakes in the opponent's team.
- $[5]$: the positions of the five beans.
- $[6]$: the full body positions of the currently controlled snake.
- $[7, 8]$: the full body positions of the other two snakes in our team.
- $[9, 10, 11]$: the full body positions of the three snakes in the opponent's team.

The features mentioned above describe the original information of the environment and is sufficient to derive information for RL policies in the game.

As for the last channel, we take the idea of rule-based method, adding the territory matrix into the feature set so that the agent is able to utilize human prior knowledge to the maximum extent. To normalize it, we divide the value of the territory matrix by the largest possible value, which is set to 240 in our method. Introducing prior knowledge helps to enhance the representations of states, thus assisting the agent to learn better and faster. The overall framework is illustrated in Figure 4.

B. Model Structure

Within the actor-critic framework for RL, we design a policy network and a value network. To be specific, integrated neural network is utilized in our design, where the policy network and value network share the main architecture that consists of eight residual blocks and each residual block includes two $3\times$

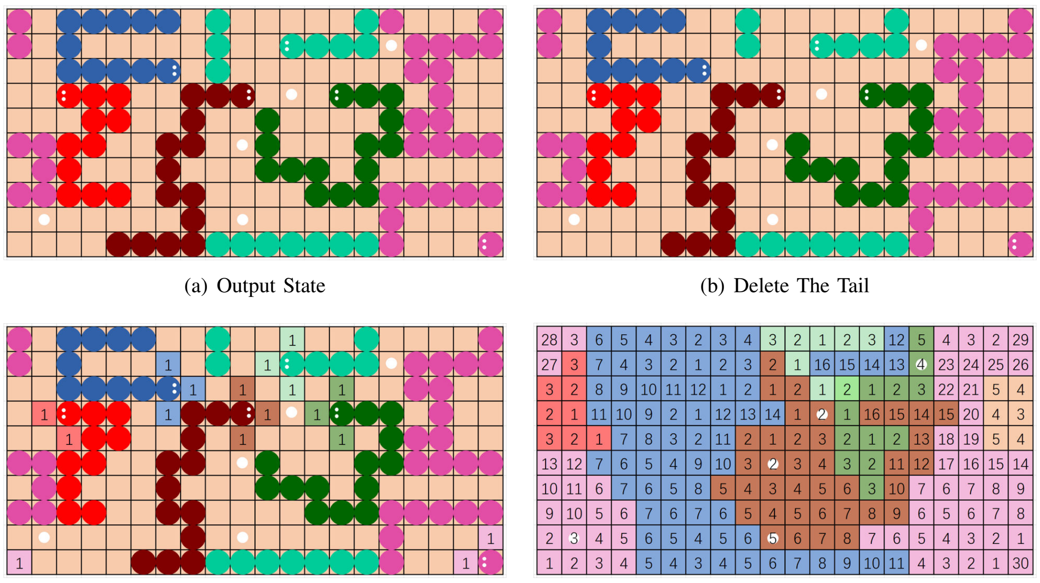

(a) Output State
(b) Delete The Tail
(c) Explore One Step
(d) Exploration Completed

Fig. 2: Calculation process of the attack range matrix. (a) is a frame of game state extracted from the game and we use it to explain how attack range matrix is obtained. The first operation is to delete the tail of the snake (remove the last body of each snake), because the position of the tail of the snake is a safe position for the head of the snake to go to in the next update step. (b) is the game state after the deletion of tails. After deletion of tails, the second operation is exploration, *i.e.*, exploring the safe locations where the snake's head can go in the next step. (c) is the matrix obtained after deletion of tails and one step of exploration, where the number 1 in the safe positions means that the snake head can reach this safe position in one step. Repeating tail deletion and exploration until every location on the map is marked, we finally obtain the attack range matrix (d).

convolution module,s. Notably, the map in 3v3 Snakes has no boundary, *i.e.*, if one snake goes out of the right side of the map, it will come in on the left, and the same goes for the upper and bottom side. To this end, we employ circular pad for edge supplement. After the extraction of features, there is a fully connected layer with a hidden size of 4, which represents four actions, for policy network, and another fully connected layer for value network, which outputs value function.

It is notable that one of the four actions in the Snakes game is always illegal. For instance, if the snake is originally moving to left, it will not be allowed to go right immediately. Therefore, we need to deal with legal action in a special way, which is described as follows:

$$P = softmax(PN(s) - (1 - l_{legal}) \times NaN), \qquad (4)$$

where l_{legal} denotes the one-hot vector for legal actions (the legal ones are 1 while illegal ones are 0), $PN(s)$ denotes the output of the policy network, and NaN denotes a very large value to mask illegal actions. After the $softmax$ operation, the probability of actions under current policy is generated.

C. Reward Design

In reinforcement learning, the reward function is used to encourage agents to act towards the expected goal. However, in our task, there exists bias between the reward function and the task target, thus making it difficult to precisely deliver our desired goal to the agents. To achieve better performance, we apply a subtly designed reward in 3v3 Snakes. We take cooperation and competition on team level into account in reward design and introduce the zero-sum condition and team reward terms in game theory. We define the initial reward r_i for i_{th} snake based on the game information as follows:

$$r_i = \begin{cases} l_n - l_{n-1}, & n < 200 \\ l_n - l_{n-1} + 10, & n = 200 \ and \ win \\ l_n - l_{n-1} - 10, & n = 200 \ and \ lose \end{cases}, \qquad (5)$$

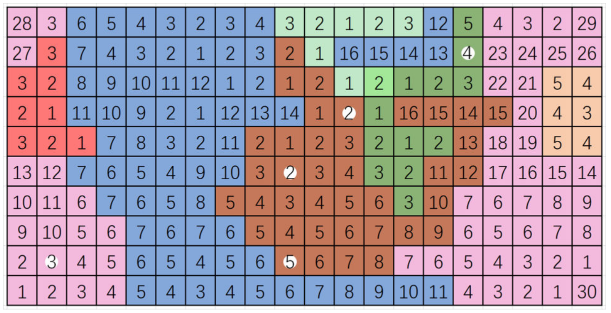
where l_n denotes the snake length of the i_{th} snake when the game proceeds to the n_{th} step, and l_{n-1} denotes the snake length of the i_{th} snake when the game proceeds to the n_{th} step.

When the game reaches the end of the two-hundred steps, we give the snake an end game reward based on the win/loss situation. Given that the team with a larger sum of snake lengths wins the game, the zero-sum reward $\hat{r}_i$ for each snake is therefore designed as:

$$\hat{r}_i = r_i - \bar{r}', \qquad (6)$$

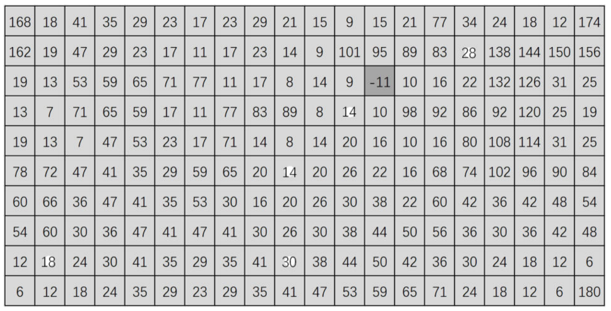
where $\bar{r}'$ represents the average value of the three snakes' reward in the enemy's team.

When deciding the action of any snake, the information from the other two snakes in the team should also be taken into account to beat the other team, which means cooperation



28	3	6	5	4	3	2	3	4	3	2	1	2	3	12	5	4	3	2	29
27	3	7	4	3	2	1	2	3	2	1	16	15	14	13	4	23	24	25	26
3	2	8	9	10	11	12	1	2	1	2	1	2	1	2	3	22	21	5	4
2	1	11	10	9	2	1	12	13	14	1	2	1	16	15	14	15	20	4	3
3	2	1	7	8	3	2	11	2	1	2	3	2	1	2	13	18	19	5	4
13	12	7	6	5	4	9	10	3	2	3	4	3	2	11	12	17	16	15	14
10	11	6	7	6	5	8	5	4	3	4	5	6	3	10	7	6	7	8	9
9	10	5	6	7	6	7	6	5	4	5	6	7	8	9	6	5	6	7	8
2	3	4	5	6	5	4	5	6	5	6	7	8	7	6	5	4	3	2	1
1	2	3	4	5	4	3	4	5	6	7	8	9	10	11	4	3	2	1	30

(a) Attack Range Matrix



168	18	41	35	29	23	17	23	29	21	15	9	15	21	77	34	24	18	12	174
162	19	47	29	23	17	11	17	23	14	9	101	95	89	83	28	138	144	150	156
19	13	53	59	65	71	77	11	17	8	14	9	-11	10	16	22	132	126	31	25
13	7	71	65	59	17	11	77	83	89	8	14	10	98	92	86	92	120	25	19
19	13	7	47	53	23	17	71	14	8	14	20	16	10	16	80	108	114	31	25
78	72	47	41	35	29	59	65	20	14	20	26	22	16	68	74	102	96	90	84
60	66	36	47	41	35	53	30	16	20	26	30	38	22	60	42	36	42	48	54
54	60	30	36	47	41	47	41	30	26	30	38	44	50	56	36	30	36	42	48
12	18	24	30	41	35	29	35	41	30	38	44	50	42	36	30	24	18	12	6
6	12	18	24	35	29	23	29	35	41	47	53	59	65	71	24	18	12	6	180

(b) Territory Matrix

Fig. 3: Attack Range Matrix and Territory Matrix. (a) is an attack range matrix. (b) is the territory matrix corresponding to (a). The coordinate point with the value of -11 is because there are two snakes whose shortest distance to that point is 2.

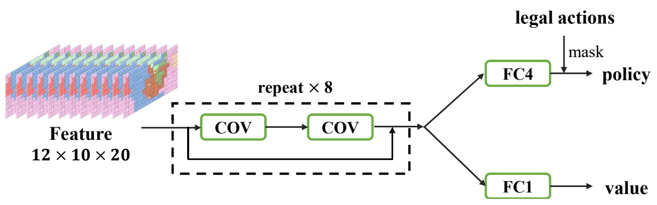


Fig. 4: The overall framework of 3v3 Snakes AI.

is a vital part of the policy. To encourage cooperation between snakes within the team, we add a cooperation term to the reward. The final reward r_i^* is thus designed as:

$$r_i^* = \hat{r}_i \times (1-\alpha) + \tilde{r} \times \alpha, \tag{7}$$

where $\tilde{r}$ is the average of $\hat{r}_i$ for the three snakes in the same team, and α is a hyper parameter to decide the weight of the cooperation term.

D. Training with Distributed PPO

In this subsection, we introduce the training process of our method. There are two kinds of modules in our training paradigm: actor, which interacts with the environment and collects training data, and learner, which trains model. In our method, we run 20 actor and 1 learner processes in parallel.

Actor The actor is responsible for the game simulation and information collection. In each iteration, each actor receives the latest model parameters from the learner. Then, the environment initializes a new episode. At each time step t of an episode, each agent receives the observation $o_t = O(s_t)$ and keeps an action-observation history τ_t. The model inputs τ_t and outputs policy $\pi_t(\cdot|\tau_t)$ and value function $v(\tau_t)$. According to π_t, agent chooses an action a_t. After the action execution, it receives a reward $r_t(s_t, a_t)$, a signal of whether episode ends $done_t$ and a transition to the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$ by the environment. At the end of the episode, actor calculates an approximate advantage function A_t for each step t, which is always estimated by Generalized Advantage Estimation (GAE) [22] as followed:

$$\begin{aligned}\hat{A}_t &= \sum_{l=1}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \\ &= \sum_{l=1}^{\infty} (\gamma\lambda)^l (r_t + \gamma V(s_{t+l+1}) - V(s_{t+l})).\end{aligned} \tag{8}$$

The agent trajectory data tuple $(o_t, a_t, \pi_t, v_t, A_t)$ is then sent to the learner to train the model. Considering self-play, each episode will produce $6 = 2 \times 3$ trajectories.

Learner The learner is responsible for network update. In each iteration, the learner first receives the collected episode data from the actors and the data is stored in a buffer. The learner then samples data from the buffer to update the network with Proximal Policy Optimization algorithm (PPO) [14]. Proximal Policy Optimization (PPO) is an effective policy gradient actor-critic algorithm, which constructs a optimization function as follows:

$$\begin{aligned}L_{clip} =& \mathbb{E}_{s\sim\rho_{old}, a\sim\pi_{old}}[\min(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A^{\pi_{\theta_{old}}}(s,a), \\ & clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, 1-\epsilon, 1+\epsilon)) A^{\pi_{\theta_{old}}}(s,a)],\end{aligned} \tag{9}$$

where ρ_{old} represents the state distribution on policy π_{old} of agent i and $A^{\pi_{\theta_{old}}}(s,a)$ is the advantage function calculated in actors. Here PPO uses the clip method to constrain the distance between the new and old policies during updating, which can effectively solve the problem that policy becomes worse in training due to an inappropriate update step size. The definition of clip function is shown in Eq. (10).

$$clip(x, x_{min}, x_{max}) = \begin{cases} x_{max}, & x > x_{max} \\ x, & x_{mix} \le x \le x_{max} \\ x_{min}, & x < x_{min} \end{cases}. \tag{10}$$

In addition to the above optimization target, PPO sets entropy of policy π as the regularization term in order to encourage exploration. The entropy loss is defined as follows:

$$L_{Ent} = \sum_{a\in A} \pi_\theta(a_t = a|o_t) \log \pi_\theta(a_t = a|o_t). \tag{11}$$

Besides, the value network is updated by:

$$L_v = \mathbb{E}_{s\sim\rho_{old}}[(V_\phi(s) - R)^2], \tag{12}$$

where $R(s) = A(s,a) + V(s)$ is the mean cumulative reward. Therefore, the total loss of PPO is formulated as follows,

$$L(\theta, \phi) = -L_{clip} - c_e L_{Ent} + c_v L_v. \tag{13}$$

where c_e is the entropy coefficient and c_v is the value coefficient.

Algorithm 1 Process of Actor

```
1:  Initialize environment ENV;
2:  Initialize model $M$ with random parameters;
3:  for Episodes=1,2,3,... do
4:      Initial state $s_0$ = ENV.reset();
5:      Set $t = 0$;
6:      while not done do
7:          for Agent=1,2,3 do
8:              $o_t^i = O^i(s_t)$;
9:              the chosen action $a_t^i$, the probability of each action
                $\pi_t^i$, value function $v_t^i$ = $M$.forward($o_t^i$);
10:         end for
11:         next state $s_{t+1}$, reward $r_t$= ENV.step([$a_t^1, a_t^2, a_t^3$]);
12:         $t = t + 1$;
13:     end while
14:     Calculate advantage function $A_t$ as Eq. (8) for each
        trajectory;
15:     For each trajectory $(o_t, a_t, \pi_t, v_t, A_t)$, save it to replay
        buffer $B$;
16:     Update model $M$ with period $I$;
17: end for
```

Algorithm 2 Process of Learner

```
1: Initialize the network parameters and replay buffer $B$;
2: for Iteration=1,2,3,... do
3:     Sample a batch of trajectory data $D = \{(o, a, \pi, v, A)\}$
       from $B$;
4:     Calculate loss $L(\theta, \phi)$ as Eq. (9), Eq. (11), Eq. (12) and
       Eq. (13);
5:     Update policy network parameters $\theta$ and value network
       parameters $\phi$ with $L(\theta, \phi)$;
6:     Send network parameters to Actor;
7: end for
```

V. Experiments

In this section, we demonstrate the effectiveness of our method with extensive experiments and build a benchmark on 3v3 Snakes as well. The code is available at https://github.com/submit-paper/3v3Snakes. These experiments are conducted to answer the following questions.

RQ1: How do different methods perform compared with humans?

RQ2: Compared to the currently strongest rule-based 3v3 Snakes algorithm, how does the performance of our RL method change with and without the introduction of the territory matrix in the fixed rules?

RQ3: What is the effect of different reward designs on the performance of the agents?

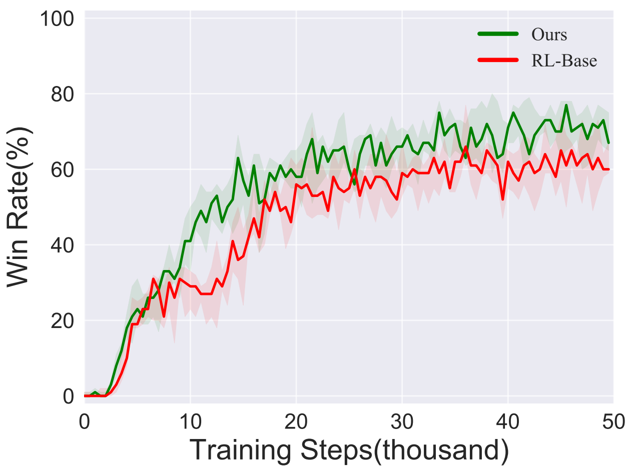


Fig. 5: Winning rate of Ours algorithm and RL-Base algorithm against **Rule** algorithm during training. The horizontal axis represents training steps of models. **Rule** is the currently strongest rule-based algorithm. **RL-Base** is the RL agent trained without territory matrix feature. **Ours** is the agent trained with ours final version of rule-enhanced RL method. Each experiment is executed 5 times with different random seeds for 250000 tests. The solid line shows the median score and the shadow area represents the standard deviation.

A. Comparison with Rule-based Algorithms

We first compete our agent with the currently strongest rule-based 3v3 Snakes AI. We demonstrate the performance of the agent with and without territory matrix from heuristic rules in feature matrix. Despite the difference in features, other hyper parameters of the two reinforcement learning methods keep the same. The models are collected and tested every 500 updates. In each test, we randomly generate one hundred initial environment states, each of which is used twice to test the method (agents from rule-based method and our method control two teams of snakes and exchange to test again with the same initial state). The experimental results are shown in Fig 5.

As shown in Fig 5, our method without the territory matrix feature is able to defeat rule-based agent with a winning rate of 50% after 20k updates and 60% when it converges. With the territory matrix information, our method is able to defeat rule-based agent with a winning rate of 50% after 15k updates and 70% when it converges. It can be seen that with the territory matrix information, the agent could achieve a better performance and require less samples.

B. Ablation Study

In this section, we analyze the impact of the zero-sum condition and the team reward term on the algorithm's performance.

3v3 Snakes is a zero-sum environment where the two teams of the game are under complete competition. To test the effect of the zero-sum reward condition on the performance of the

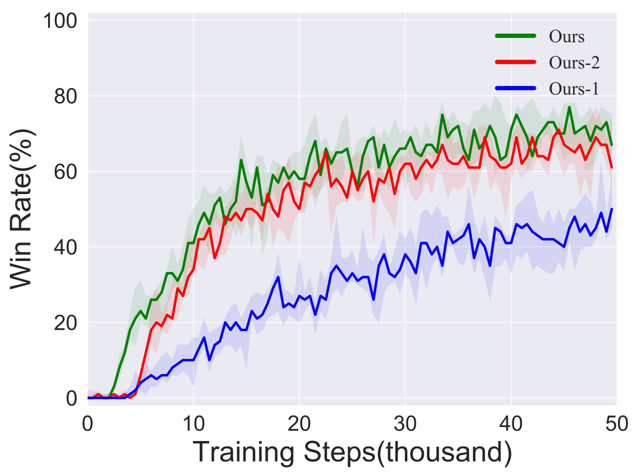


Fig. 6: Winning rate for our algorithm and ablations against rule-based method. The horizontal axis represents training steps of models. **Ours-1** is the RL agent without satisfying zero-sum condition. **Ours-2** is the RL agent without the team reward term. Each experiment also is executed 5 times with different random seeds for 250000 tests. The solid line also shows the median score and the shadow area represents the standard deviation.

agent, we keep the team reward term but disregard the zero-sum condition. A non-zero-sum reward function is designed while keeping the other experimental settings the same. The non-zero-sum reward for each snake is designed as:

$$r_i^* = r_i \times (1 - \alpha) + \ddot{r} \times \alpha, \tag{14}$$

$\ddot{r}$ represents the average of the r_i of the three snakes in the same team. The reward function here does not include the changes in length of the enemy snakes.

In addition, in 3v3 Snakes, one team controls three snakes at the same time. In order to achieve victory for the whole team, the information of the other two snakes should be very important when deciding the action of the snake. In this section, we introduce the team reward to include the information of the other snakes. To test the effect of the team reward term, we maintain the zero-sum condition but discard the team reward term. The devised reward function without team reward term for each snake is then designed as:

$$r_i^* = \hat{r}_i. \tag{15}$$

With the rest of the experimental settings the same, we test the performance of the two devised reward functions. Using different reward functions, we separately test the agent's winning rate against the rule-based agent during the training process. The experimental results are shown in Fig 6.

It's observed that without zero-sum condition or the team reward term, the performance of the agent both drop significantly. Without team reward term, the RL method requires roughly 18,000 updates to reach a comparable performance of the rule-based agent, which is 20% slower than the original algorithm. The final winning rate also drops to 65%. Disregarding zero-sum condition, both the training speed and the best performance will be badly influenced. After 50,000 updates, the agent is still not able to match the rule-based agent. This shows that both the team reward term and the zero-sum condition are essential parts of our RL method.

We conducted 15 combat experiments between six versions of algorithms to test the winning rates against each other.

The chosen algorithms are shown as below:

- Random: A random strategy algorithm.
- Rule: The currently strongest rule-based algorithm in JIDI platform. [2]
- RL-base: The RL agent without territory matrix feature.
- Ours-1: The RL agent without satisfying zero-sum condition.
- Ours-2: The RL agent without the team reward term.
- Ours: Our final version of rule-enhanced RL method (combining all the elements above).

All RL agents use models trained after 50,000 updates. Tests are conducted between agents from every two algorithms. In each test, we randomly generate 1000 initial environment states. Starting from each initial state, the teams will play against each other twice, similar to the test as previously mentioned.

The results are shown in Table I. We counted not only the win rate of the AI on both sides of the match, but also the average leading length of the snake body per game and the percentage of the average leading length of the snake body per game to the average total length of the snake body per game.

Table I shows that the Random strategy has barely any chance to beat the remaining five AIs. Three of the four RL AIs have a winning rate of over 60% against the currently strongest Rule-based AI. Only the fourth AI disregarding the zero-sum condition has a winning rate below 50% against the rule-based AI. Our AI has achieved a 71.9% win rate against Rule based AI, with an average lead of 12.79 snake lengths per game. In a game map of size 10×20, this lead is very large. Our AI also achieves a 62% win rate against the RL-Base, with an average lead of 7.01 snake lengths per game, which shows that the introduction of the territory matrix information in the feature vector effectively helps to improve the game performance.

To better compare the performance of the six AIs, we also analyse the percentage of the average leading length of the snake body per game to the average total length of the snake body per game. This metric reached 28.02% when Ours is playing against Rule, meaning that Rule's average total snake body length is only 71.98% of Ours' average total snake body length.

C. Comparison with Human Players

To further estimate the performance of the Ours agent, we conduct a set of tests between the agent and human players. In

[2]https://github.com/CarlossShi/Competition_3v3snakes

A \ B	Random	Rule	RL-Base	Ours-1	Ours-2
Rule	100% / 42.15 / 81.53%	——	——	——	——
RL-Base	99.9% / 56.55 / 85.63%	62.8% / 7.49 / 17.39%	——	——	——
Ours-1	99.95% / 50.09 / 84.01%	47.6% / 0.09 / 0.23%	29.65% / -12.41 / -32.41%	——	——
Ours-2	100% / 55.02 / 85.34%	67.45% / 10.29 / 23.24%	53.75% / 3.29 / 7.09%	72.35% / 14.34 / 27.98%	——
Ours	**100% / 57.10 / 85.75%**	**71.9% / 12.79 / 28.02%**	**62% / 7.01 / 14.93%**	**78.8% / 18.96 / 35.35%**	**58.4% / 5.17 / 11.18%**

TABLE I: The average performance of the compared algorithms by playing 2000 episodes of 3v3 Snakes. In the table, the first column in the table denotes the winning rate of algorithm A against algorithm B. The second column indicates the average sum of length of algorithm A when the game terminates and we use l_A to represent it. The third column represents the proportion of length of algorithm A exceeding over algorithm B compared to the length achieved by algorithm A, which can be written as $(l_A - l_B)/l_A$.

the test, human players are all extremely familiar with the 3v3 Snakes game. The AI fights three agents at the same time and each human player control one snake. After extensive testing statistics, the Ours gets a winning rate of 63.75%, with an average lead of 11.4 snake lengths per game over the human players. This result demonstrate that our AI has reached the super-human level.

VI. Conclusion

This work presents a strong AI program for 3v3 Snakes and builds a benchmark for this game, which will be utilized for future research in the community. To address the challenges including the huge state space and the involvement of both cooperation and competition in 3v3 Snakes, we propose a rule-enhanced multi-agent reinforcement learning algorithm. Extensive experimental results reveal that our AI outperforms the rule-based algorithms and even exceeds human level, which demonstrate the effectiveness and superiority of our method. In our future work, we will leverage our idea to more complex multi-player games and explore better solutions to game AI.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] T. Shu, J. Liu, and G. N. Yannakakis, "Experience-driven pcg via reinforcement learning: A super mario bros study," in *IEEE Conference on Games (CoG)*, 2021, pp. 1–9.

[3] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, and H.-W. Hon, "Suphx: Mastering mahjong with deep reinforcement learning," *arXiv preprint arXiv:2003.13590*, 2020.

[4] N. Brown and T. Sandholm, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[5] D.-W. Kim, S. Park, and S.-i. Yang, "Mastering fighting game using deep reinforcement learning with self-play," in *Proceedings of the IEEE Conference on Games (CoG)*, 2020, pp. 576–583.

[6] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The StarCraft Multi-Agent Challenge." *CoRR*, vol. abs/1902.04043, 2019.

[7] J. Bishop, J. Burgess, C. Ramos, J. B. Driggs, T. Williams, C. C. Tossell, E. Phillips, T. H. Shaw, and E. J. de Visser, "Chaopt: a testbed for evaluating human-autonomy team collaboration using the video game overcooked! 2," in *Proceedings of the Systems and Information Engineering Design Symposium (SIEDS)*, 2020, pp. 1–6.

[8] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[9] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[10] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo *et al.*, "Mastering complex control in moba games with deep reinforcement learning." in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[11] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv: Learning*, 2017.

[15] T. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[16] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[17] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[19] Z. Wei, D. Wang, M. Zhang, A.-H. Tan, C. Miao, and Y. Zhou, "Autonomous agents in snake game via deep reinforcement learning," in *Proceedings of the IEEE International Conference on Agents (ICA)*, 2018, pp. 20–25.

[20] S. Sharma, S. Mishra, N. Deodhar, A. Katageri, and P. Sagar, "Solving the classic snake game using ai," in *Proceedings of the IEEE Pune Section International Conference (PuneCon)*, 2019, pp. 1–4.

[21] A. J. Almalki and P. Wocjan, "Exploration of reinforcement learning to play snake game," in *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019, pp. 377–381.

[22] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.