Large Scale Deep Reinforcement Learning in War-games

Hanchao Wang
College of Intelligence and Computing
Tianjin University
Tianjin, China
ares1899@126.com

Hongyao Tang
College of Intelligence and Computing
Tianjin University
Tianjin, China
bluecontra@tju.edu.cn

Jianye Hao
College of Intelligence and Computing
Tianjin University
Tianjin, China
haojianye@126.com

Xiaotian Hao
College of Intelligence and Computing
Tianjin University
Tianjin, China
15122938076@163.com

Yue Fu
College of Intelligence and Computing
Tianjin University
Tianjin, China
fuyue@tju.edu.cn

Yi Ma
College of Intelligence and Computing
Tianjin University
Tianjin, China
mayi@tju.edu.cn

2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 978-1-7281-6215-7/20/$31.00 ©2020 IEEE DOI: 10.1109/BIBM49941.2020.9313387

***Abstract*—War-game is a type of multi-agent real-time strategy game, with challenges of the large-scale decision-making space and the flexible and changeable battlefield situation. In addition to the military field, it has played a role in fields including epidemic prevention and pest control. In recent years, more and more learning algorithms have tried to solve this kind of game. However, the existing methods have not yet given a satisfactory solution for the wargame, especially when preparation time is limited. In this background, we try to solve a traditional war-game based on hexagon grids. We propose a hierarchical multi-agent reinforcement learning framework to rapidly training an AI model for the war-game. The higher-level network in our hierarchical framework is used for task decision, it solves the credit assignment problem between agents through cooperative training. The lower-level network is mainly used for route planning, and it can be reused through parameter sharing for all the agents and all the maps. To deal with various opponents, we improve the robustness of the model through a grouped self-play approach. In experiments, we get encouraging results which show that the hierarchical structure allows agents to learn their strategies effectively. Our final AI model demonstrates that our methods can effectively deal with the challenges in the war-game.**

***Index Terms*—Multi-agent system, Reinforcement Learning, War-game, Self-play**

I. Introduction

War-game is a general term for a large class of real-time strategy games that aimed at simulation deductions under complex environmental changes to achieve environment recognizing, auxiliary decision-making, and combat theory innovation [1]. Since the war-game platform can be used as an effective tool to study the possibility of developments of conflict events, it has been widely used in the military field for many years, and even spread to the fields of epidemic prevention and pest control. For example, in dealing with classical swine fever virus (CSFV), war-game is used to simulate the spread of CSFV and also the influence of different countermeasures.

There are many pieces of research on intelligent decision-making in war-games, however, it is still an open question in the Artificial Intelligence (AI) field. A few previous works propose traditional solutions such as heuristic approaches. Reference [1] focuses on the question of finding good representations for the AI design. Reference [3] is based on the partially automated use of the rules of the game, as well as some common sense and historical military knowledge, to design relevant heuristics. Recently, a few works demonstrate success of Reinforcement Learning (RL) with Deep Neural Networks (DNNs) in solving complex video games, e.g., Atari [5], the game of Go [6], Dota 2 [7] and StarCraft [8]. However, to the best of our knowledge, how a learning agent can perform in a war-game is still unexplored.

In this paper, we study the discrete space war-game in hexagonal maps and takes the lead in proposing a formal description of this kind of war-game as a partially observable Markov game. We analyze the difficulties and challenges of learning an effective agent and then propose a task-behavior based hierarchical multi-agent reinforcement learning framework. Our proposed learning framework contains a hierarchical network architecture with higher-level cooperative training for tasks and lower-level independent learning for task-associated behaviors. After a warm-up training against rule-based AI, our AI models also have a grouped self-play approach to ensure their effectiveness and robustness against various opponents. Moreover, other mechanisms like policy distillation, curriculum Learning, parameter sharing, and prior knowledge integration are also adopted to facilitate the learning process.

To evaluate our framework, we conduct a variety of experiments in given war-game scenarios. The results show that our approach successfully learns an effective AI model and outperforms representative benchmarks. Moreover, we briefly discuss the limitation of our methods and open questions in the war-game, opening up a few directions for future research.

II. War-Game on Hexagonal Grids

In this section, we introduce the war-game in detail and discuss its characteristics and main challenges.

A. Game Elements

War-game is played on a map with a matrix of hexagonal grids. In this game, players from two camps control their own battle pieces to fight for control points and a higher score.

The hexagonal grid (called *grid* for short) is the smallest unit of the map of war-game, as shown in Fig. 1. Each grid contains two kinds of information: *location coordinate* and *grid properties*. In the game, the location coordinate of a grid is represented by a four-digit positive integer, with the first two-digit represent the index of abscissa, and the last two represent the index of ordinate. For a grid, its properties include its altitude and a bool vector representing its type. The bool vector shows whether the grid is controllable, passable, paved, concealed, and slushy.

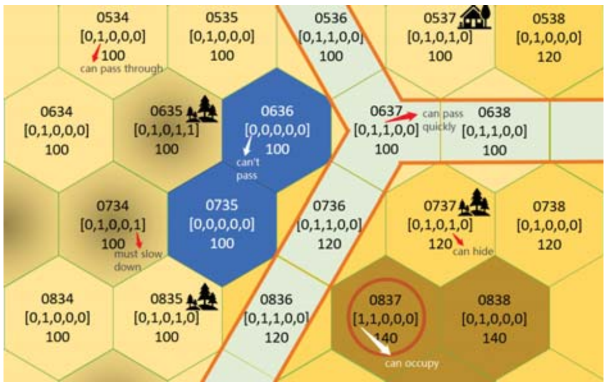

Fig. 1. Hexagonal grids.

The battle piece (called *piece* for short) is the operational object that players control in the game. There are four types of pieces, including *tank*, *chariot*, *soldier* and *aerocraft*. The pieces have their own properties and mutual properties, as shown in Fig. 2. Their own properties are divided into intrinsic properties and dynamic properties, the former including *type*, *camp*, *value* and *speed*, and the latter including *visible*, *position*, *blood* and *cold-down*(C.-D.).

Their mutual properties contain two types of interactions: *attack effect* and *visible to*. For an attacker i_{A} and a defender i_{D}, The attack effect F_{Att} is related to their distance Δx_{AD} and their altitude difference Δh_{AD}. And whether i_{D} is visible to i_{A} depends on their positions $\{x_{\mathrm{A}}, x_{\mathrm{D}}\}$.

The goal of the game is to get a higher score, which is the sum of values of all its pieces including the control points it occupies. For each piece p, suppose that its value is v_p, its blood is n_p, and its camp is $c_p \in \{\text{"Blue"}, \text{"Red"}, \text{"Green"}\}$. Formally, the optimization goal for camp c is (1):

$$\text{Maximize} \quad v_c = \Sigma_p^{c_p=c}(v_p \cdot n_p) - \Sigma_p^{c_p \neq c}(v_p \cdot n_p) \qquad (1)$$

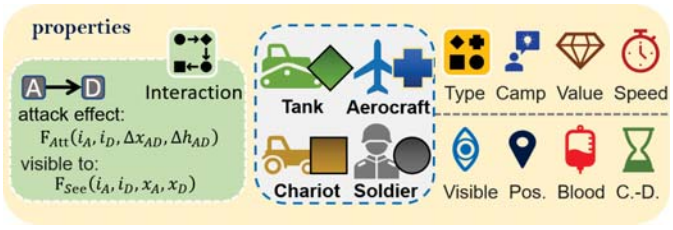

Fig. 2. Battle pieces.

B. Game Challenges

To win the match of war-game, the design of AI must take into account the challenges followed:

- **Large state space**. War-game is played on a large map containing near 5000 grids. Each player side has at least 6 pieces. Observation and state space for each piece is high-dimensional.
- **Intricate action space**. Each piece has about 11 kind of actions, including *null*, *move*, *hide*, *occupy*, *shoot*, *guide-shoot*, *indirect-shoot*, *get-on*, *get-off*, *decompress* and *stop*. Some of these actions have lots of sub-parameters, whose size is even variable. The semantic relationship between those actions is also complicated.
- **Partially-observed state**. Agents require making inferences based on incomplete information, and modeling and predicting the opponent's behavior.
- **Instability of state transition**. In the game, a player can't know what the opponent is going to do, which will cause the uncertainty of state transition. The multiple random corrections of results after shooting in the environment also complicate the state transition.
- **Tactical changes of the opponent**. Considering the intentions of opponents in the war-game may change at any time, a good player needs to learn to deal with multiple types of opponents.
- **Long time horizon**. In our simplified competition of war-game, there are 1800 steps in each episode to make decisions.
- **Multiple factors**. Any result in the war-game is caused by multiple factors, including all the operations of the player and its opponent, and also the random factors of the environment. Credit-assignment of the reward to all the pieces is difficult.
- **Knowledge usage**. Training an agent from zero requires a huge time overhead. How to better combine the existing expert knowledge is the main problem to complete the model training within a limited time.

III. Approach

In our approach, we first formally model the war-game process, especially for the action space. Secondly, we propose a hierarchical framework corresponding to the action space. Finally, we give RL training methods for our framework, and mention other auxiliary methods to improve the RL agents.

A. Problem Statement

Uniform definition. Since the war-game can be regarded as a fully cooperative Markov game with a partially observable environment, we take each piece in this game as an RL agent and draw lessons from the definition of Decentralized Partially Observable Markov Decision Process (Dec-POMDP).

Formally, we define the war-game as a multi-agent system consisting of $(N + M)$ agents, which are divided into two camps $\mathcal{I} = \{1, 2, \ldots, N\}$ and $\mathcal{I}^- = \{-1, -2, \ldots, -M\}$. In each step with the global state $s \in \mathcal{S}$, each agent $i \in \mathcal{I} \cup \mathcal{I}^-$ gets its own observation o_i from the environment and make an action a_i according to its decision model. The environment gets the joint actions from two camps $\boldsymbol{a} \equiv \{a_1, \ldots, a_N\} \in \mathcal{A}$, $\boldsymbol{a} \equiv \{a_{-1}, \ldots, a_{-M}\} \in \mathcal{A}^-$ and then figures out the next global state $s' \in \mathcal{S}$ according to the transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{A}^- \to \mathcal{S}$, and the reward r calculated by the difference between the two states s, s'.

States and Observations. We formally divide the global state as $s := \{\sigma_E, \sigma_1, \sigma_2, \ldots, \sigma_N, \sigma_{-1}, \ldots, \sigma_{-M}\}$, where σ_E is the map information and each σ_i is the piece information of each agent $i \in \mathcal{I} \cup \mathcal{I}^-$. The map information includes the grid properties of all its grids and the topological relations of those grids. The piece information includes the piece's own properties mentioned before.

In one side $\mathcal{I}$, the observation o_i for agent $i \in \mathcal{I}$ is defined as $o_i := \{\sigma_i, \sigma_E, \sigma_1, \ldots, \sigma_N, u(\sigma_{-1}, t_{-1}), \ldots, u(\sigma_{-M}, t_{-M})\}$, where $u(\sigma_j, t_j)$ is a supplement function for the intermittent information of the enemy agent $j \in \mathcal{I}^-$, and t_j indicates that agent j has been invisible for t_j steps. The joint observation of this side is $\boldsymbol{o} \equiv \{o_1, \ldots, o_N\} \in \mathcal{O}$. Since the observation in war-game contains the available information completely, we use $\boldsymbol{o}$ as a basis for decision making rather than the history information $\boldsymbol{\tau}$ defined in Dec-POMDP.

Action. We redefined the 11 kinds of original actions in a task-associated hierarchical action model with two levels.

We define the higher-level action as the task selection, which has three types:

- *The grid-based tasks*. In this kind of tasks, an agent first selects a grid from the available target sets, which is designed artificially based on the information of the environment, and then perform a task related to that grid. The task is usually to move to the corresponding grid and then perform some discrete actions, such as *stop*, *get-off*, *occupy*, or *hide*.
- *The piece-based tasks*. In this kind of task, an agent selects an opponent's piece and performs a task related to that piece. The task is usually to move to a suitable grid for shooting the piece and then perform some discrete actions, such as *stop* (shadow) and *shoot*.
- *Stop*. It means to terminate the ongoing action of the agent or just stay if there is no ongoing action.

Each step for agent $i \in \mathcal{I}$, its action g_i in higher-level is to select a task from an available task set G_i, where the set $G_i = \{g_i^{(1)}, \cdots, g_i^{(K_i)}\}$ with K_i tasks is generated by rule from all the above tasks mentioned, according to its individual observation o_i.

We define the lower-level action as the behavior that mainly serves route planning. An action d in lower-level mainly gives a move direction: $d \in \{\odot, \leftarrow, \swarrow, \searrow, \rightarrow, \nearrow, \nwarrow\}$. For each agent $i \in \mathcal{I}$, there is a action generation rule ϕ_i: if the lower-level action d_i is some kind of arrow, then it will move in that direction; if "$d_i = \odot$", then it should stop move and perform the action according to its higher-level task g_i. Formally, an action a_i of agent i is decided by (2):

$$a_i = \phi_i(g_i, d_i) = \begin{cases} a(g_i) & d_i = \odot \\ d_i & \text{otherwise} \end{cases} \tag{2}$$

Transition. Since the strategies of opponents is unknown, we regard the distribution of the opponents' strategies as a part of the environment's transition. The new transition function is $T^+ : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$.

Reward. The overall reward r_Σ with the MDP transition $\langle s, \boldsymbol{a}, s' \rangle$ is figured by the difference between the states' scores v_c (defined in (1)):

$$r_\Sigma = v_c(s') - v_c(s) \tag{3}$$

In step t, considering globle states s^t, $s^{(t+1)}$ are distributions on the the joint observation $\boldsymbol{o}^t$ and the joint action $\boldsymbol{a}^t$, the object function to optimize is (4):

$$J = \mathbb{E}[\sum_{t=0}^{\infty} \left(R_\Sigma(\boldsymbol{o}^t, \boldsymbol{a}^t)\right)] \tag{4}$$

where $r_\Sigma \sim R_\Sigma(\cdot)$ is mainly provided for higher-level task selection.

Simplifications. Our method has some simplifications in war-game to simplify the environment:

- **Known Environment**. Known maps and fixed starting point (initial global stste) for each map.
- **State representation**. For the positional information of a location, we use the property of the 6 grids nearby instead of using the information of the whole map.
- **Reduced available action set**. For the first kind of higher-level task, we only choose some important grids as the option to deal with the large scale problem.

B. Networks Architecture

One characteristic of war-game is its complex action space, and the atomic action decision model from most RL algorithms can not adapt to this environment. To tackle this problem, we propose an interactive framework according to the statement before, shown in Fig. 3, with task-behavior based hierarchical decision networks for agents to take composite actions with policies at different levels. A similar idea is adopted in solving parametrized action problems. [18].

The hierarchical decision network consists of two levels as shown in Fig. 4 (a), where a higher-level network, also called *upper policy*, selects an available task for the agent and a lower-level network, called *lower policy*, selects a task-associated action to perform that task if necessary.

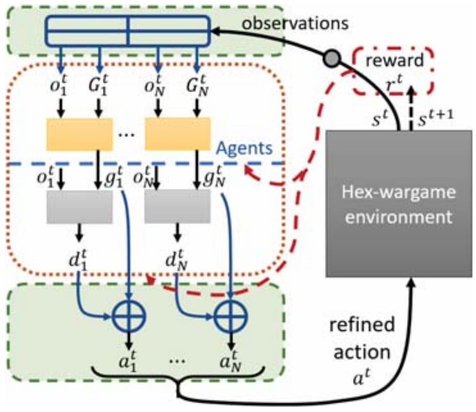

Fig. 3. An interactive framework with hierarchical architecture.

Higher-level Network (Upper Policy). The structure of a higher-level network is illustrated In Fig. 4 (b). The upper policy of the agent i takes its observation o_i and available tasks set $G_i = \{g_i^{(1)}, \cdots, g_i^{(m)}\}$ as input. The feature $h_i^{(k)}$ of task $g_i^{(k)}$ with observation o_i is obtained through the embedding module, and then all the features are aggregated to get the representation h_Σ of the entire task set G_i. Finally, according to o_i and the h_Σ of all the tasks, the value $Q_i^{(k)}$ of task $g_i^{(k)}$ is obtained through the estimating module.

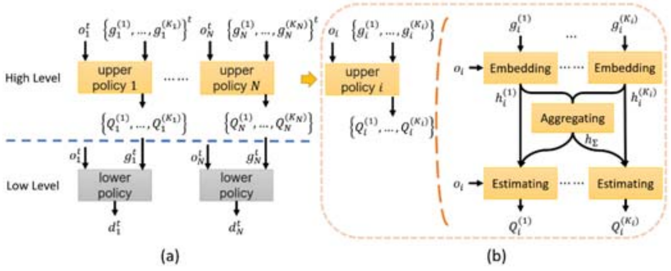

Fig. 4. The hierarchical networks with the details of the upper policy.

Lower-level Network (Lower Policy). In the low level, according to the state and the task given by upper policy, the lower policy chooses an associated direction d_i. In our opinion, the lower policies for all the agents are relatively independent with each other since the degree of cooperation relies more on higher-level task decisions. One advantage of hierarchical architecture is the disentanglement and flexibility of composite actions in training and execution. In some cases, we can replace the lower-level network with various methods from prior knowledge, e.g., heuristic methods or offline-learned models.

Interaction with the war-game. In real platform, human players use the keyboard and the mouse to control the pieces and use the computer monitor to get the environment information. In contrast, the war-game agents in Fig. 3 get the structural state information with available actions (tasks) from the environment and send command messages to execute the joint action $\boldsymbol{a}$ in the platform. The agents also need to figure out the reward according to the structured message from the environment.

For the states as the input of our network, we extract the required elements from the structural message as the states defined in the MDP problem mentioned before. For generating actions to execute in the environment, we use deterministic rules $\phi(\cdot)$ defined in (2).

C. Training

Our training process of the multiagent hierarchical architecture proposed in the previous subsection contains four aspects in the following. First, we preprocess the map information to deal with the large state space. Second, we train the upper policy through Q-Mix architecture [19] for the emergence of task-level cooperative policies. Third, we use parameter sharing for lower policies to accelerate lower-level training. Finally, we resort to a grouped self-play approach to further improve the learning agent for effectiveness and robustness when playing with multi-mode opponents.

Preprocessing. Before training our network, we preprocessed the information on the map. First, we counted the frequency of each agent appearing in each grid of the map, according to the historical human player data. The union of two "control point" grids and the top k grids, with the frequency we counted from high to low, are selected as the basic collection $G^L(k)$ representing the location-based tasks. Secondly, we extract the altitude differences of the grids for the boundary identification.

Cooperative Training in Higher-level. To facilitate the coordination of multiagent, we focus the cooperative training in the aspect of tasks, i.e., upper policies. Therefore, we use Q-Mix architecture for higher-level training which is widely demonstrated to effectively facilitate the emergence of multiagent cooperation. An illustration of Q-Mix architecture is shown in Fig. 5, where a mixing network (in green) takes higher-level action values $\{Q_i\}_{i=0}^N$ from each upper policy as inputs and mixes them monotonically, producing the joint value Q_Σ. This ensures that a global *argmax* operation performed on Q_Σ yields the same result as a set of individual *argmax* performed on individual Q_i. The parameters of the mixing network are constrained to be non-negative to ensure monotonicity mentioned above and are produced by separate Hyper-networks [22] (the red part of Figure 5), which takes the global state s as input. The Q-Mix architecture allows agents to train their individual Q-network (upper policy) centrally through updating a joint action-value function, inducing more efficient updates towards coordination. With global information fully utilized during training, the execution of each agent remains decentralized depending only on local information (with its individual upper policy).

Independent Training in Lower-level. The lower policy of each agent is learned with the DQN network with no sophisticated architecture. The reward function for Bellman updating is specially designed according to the route planning problem:

$$r_L = r^{\text{task}} + \alpha \cdot r_\Sigma - \beta \tag{5}$$

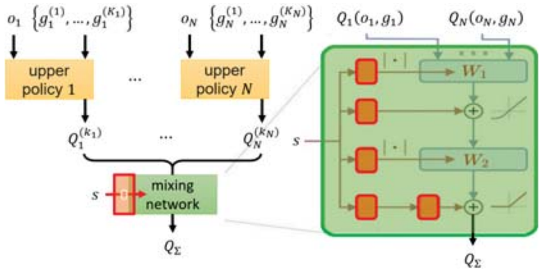

Fig. 5. The training archetecture of Q-Mix.

where r^{task} is 1 if the given task is finished (otherwise it is 0), r_Σ is the overall reward mentioned before, $\alpha \in \mathbb{R}^+$ is a constant coefficient of r_Σ, $\beta \in (0,1)$ is a constant negative feedback. Moreover, we use parameter sharing for lower-level learning for all the agents.

Alternate Training. The training of the upper policy and the lower policy in our hierarchical architecture is carried out alternately. When training the upper policy or the lower policy, we fix the parameters of the other. We train the upper policy at the beginning, with a heuristic lower policy before the lower-level network is well trained.

Policy Evolution through Grouped Self-play. Considering the diversity of opponents, traditional self-play methods cannot ensure a policy covering all opponents as they just extremely acclimatize to the latest opponents. After the warm-up training against some rule-based AI, our AI models will further enhance their ability by playing with themselves.

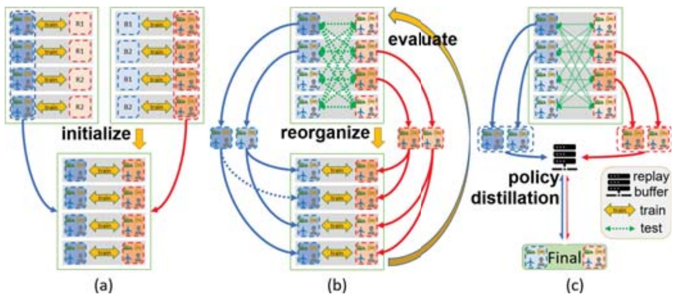

Fig. 6. The complete process for the grouped self-play approach.

As illustrated in Fig. 6 (a), we obtain many AI models after warm-up training in different random processes and group them for the self-play approach. As shown in Fig. 6 (b), the self-play approach has many iterations to facilitate the evolution of the models. In each iteration, models are first trained with each other in groups for a period of time. Then a single round-robin competition is conducted among the models, and the average win-rate is calculated to evaluate each model. For both sides (i.e. blue and red) of models, we screen several models with higher win-rate and duplicate them to form the new generation for the next iteration. At the end of the grouped self-play approach, we combined the top policies through *policy distillation* [20], shown in Fig. 6(c).

Other tricks for training based on human knowledge. In addition to the module division in the hierarchical framework and the self-play technique in policy evolution, we also used some prior knowledge for agents training as shown below.

- **Well-designed Decision Points**. Considering that each agent needs to cool-down to perform certain actions after its last action, making decisions too frequent can backfire. Our agents make decisions by fixed number of steps Δt while both the position and blood of all the agents are unchanged between two adjacent decision time points. In our experiments, $\Delta t = 75$ as the minimum common divisor of all kinds of cold-down intervals. This trick solves the above problem and does not affect the optimality of the final policy. Furthermore, controlling the decision frequency contribute to the reward allocation over a long time horizon.
- **Warm start** for training. Since the initial state of all agents in the original environment is fixed, the exploration in such a large state space is not effective. To solve this problem, we first take the convex hulls G^B of all the grids in G^L as the boundary of the battle zone. Similar to [21], at the beginning of each training period, each agent first randomly selects a grid in G^B to move and starts making decisions until reached the grid. This trick increases the frequency of exploration near critical states in the self-play period.

IV. Experiments and Evaluations

In this section, we describe our experimentation with general settings and give comparisons and analysis of the results in different aspects, including the score and win-rate, the selection ratio of different tasks, the target assignment, and the performance of the lower policy. The results show the effects of various method designs and some reasonable strategies of our AI model.

For general settings, our methods mainly adopt GPU training: the training process was conducted on 4 Nvidia-2080Ti. Our training environment adopts a simple hex-war platform provided by a team of CASIA[1]. We play 16 games at a time on the platform for training. Each game has 1800 steps in an episode, with about 10 seconds time interval. By default, each basic network module is instantiated by a multi-layer perceptron (MLP) with 2 groups of linear layer and ReLU. For training parameters, learn rate $\text{lr} = 0.1$ and update timestep $\text{a} = 0.1$. All the models are sampling in the ϵ-greedy way, where $\epsilon = 0.5$ at the beginning and dropped by 0.01 every 100 episodes until $\epsilon = 0.05$. The reward r_Σ for the upper policy updating is figured our by (3). The reward r_L for the lower policy updating is figured out by (5), where $\alpha = 0.01$ and $\beta = 0.05$.

A. Performance

After a month of intensive training, our final AI model beat the previous models to varying degrees, and we won the

[1]*The Institute of Automation, Chinese Academy of Sciences

meritorious winner prize in the final competition against 32 professional teams. To show more performance, we compared the win-rate and scoring curve during the training period.

Firstly, we trained our AI on three built-in maps, against two rule-based AI models called ”MA1” and ”MA2” for the warm-up training. Both the manuscript AI designed by experts has good performance against amateur players. One of them is more offensive, the other more defensive. With the same relevant training parameters, the comparison of learning curves of our methods (without the grouped self-play approach) called “HRL” and of deep Q-learning method by treating the joint action as one action called “DQN” is shown in Fig. 7

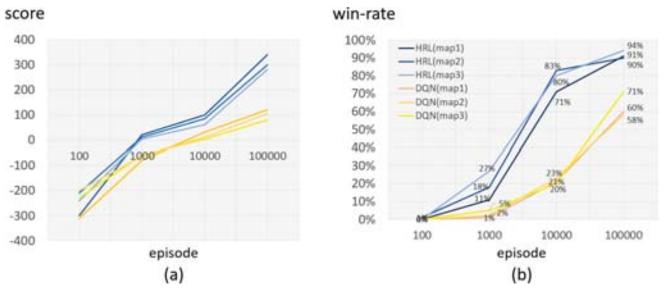

Fig. 7. The test score and win-rate of our models compared with DQN.

Fig. 7 shows that our methods can quickly (with about 10000 episodes) obtain a solution to defeat the opponent and our models converge to a higher level in win-rate at the end of the training. The overall performance of our method is better than DQN methods. The analysis believes that due to the flat action space settings in DQN, the candidate actions with different semantic mixed together can reduce sampling efficiency in training.

Secondly, for improving our AI to adapt to diversified opponents, we adopted the grouped self-play approach with 16 groups for training and retain 4 red models and 4 blue models in every self-play iteration. We kept the strongest group of models of both red and blue sides after warm-up training against “MA1” and “MA2”, called “HRL0”. After a self-play approach for 10 iterations, we select the top 2 groups of AI models of both sides, called “HRL1”, “HRL2”. Then we combine “HRL1” and “HRL2” by policy distillation and get the final AI model, called “CAI”. Without loss of generality, the performance (win-rate) of different AI models on the first map is shown in Tab.I.

TABLE I
WIN-RATE FOR THE AI MODELS

Win-rate for Red‖Blue	Opponent AI			
	MA1	*MA2*	*HRL0*	*HRL1*
HRL0	94%‖93%	89%‖87%	53%‖47%	20%‖23%
HRL1	97%‖96%	91%‖94%	77%‖80%	51%‖49%
HRL2	90%‖93%	94%‖95%	78%‖78%	46%‖48%
CAI	100%‖100%	99%‖95%	88%‖85%	69%‖64%

Tab.I shows that the method of grouped self-play can effectively improve the win-rate. And the combination model by policy distillation can maintain a higher win-rate against each opponent.

B. Higher-level task decision

At the higher-level task decisions, whether to deploy or strike against visible opponents is a common question. To facilitate the narrative, we artificially divide this game into deployment phase, preemption phase, and endgame phase. We show the results of task decisions by the model “CAI”:

- **Deployment phase** refers to the period from the beginning of the game to the time that opponents are mostly visible. In this phase, agents basically select the grid-based tasks and to deploy.
- **Endgame phase** refers to the period from the time that an agent occupies the control point. And it lasts until the end of the game. In this phase, all agents basically select the piece-based tasks.
- **Preemption phase** refers to the period between the two phases mentioned above. In this phase, the chariot and areocraft tend to choose from the piece-based tasks at the higher-level.

C. Higher-level target selection for shooting

In a game, for changing the situation, agents need to select opponent pieces to shoot. We have counted the target pieces selection of “CAI” in all the shooting situations and got the following conclusions:

- The agents tend to choose a target with lower blood.
- The agents tend to choose a target with a higher score.

More generally, for each type of agent of both red and blue sides of “CAI”, we calculated the target selection frequency in type in the first half of the game, as shown in Fig.8.

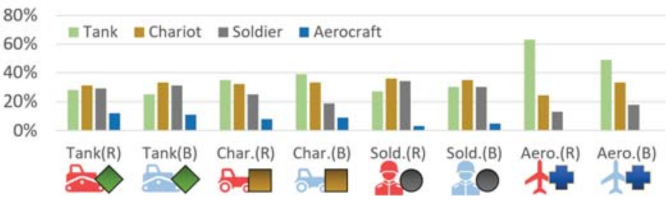

Fig. 8. The distributions of the target piece selection in different types

In Fig.8, the selection by the tank is balanced, which is consistent with its flexible combat capabilities and its strong strike capability for all kinds of targets. And the aerocraft is more likely to shoot the tank and less targeted. Analysis believes that the target selection is based on two considerations, one is to gain more score immediately, the other is to early weaken the opponent’s strength to avoid losing the score in the future.

D. Lower-level movement

In our final model, the move actions generated by the lower policies of “CAI” are more than 80% consistent with those generated by the heuristic rules designed by experts. The lower-level movement in the replay is basically reasonable.

V. Conclusions and Future Work

This paper proposed a hierarchical framework for the multi-agent war-game, which learns both its upper policy and lower policy by reinforcement learning. It took three months to design our AI for the game and test on the maps. Our model can cope with most of the challenges in the war-game, and it has achieved good results in experiments. The success of our method in war-games suggests that general-purpose machine learning algorithms may have a substantial effect on complex real-world problems.

However, our methods still need further improvement. In the future, we will explore how to solve the reward assignments among the agents and among the time horizon. We will try to incorporate graph network methods and distributed representation methods to solve this problem. We will also do further research on the structural information of the war-game, especially the processing of hexagon grids on the map. We also hope that the behavior of future models can give us some inspiration about the war-games.

References

[1] V. Corruble, C. A. Madeira, and G. L. Ramalho, "Steps toward building of a good ai for complex war-game-type simulation games," in GAME-ON, 2002.

[2] B. L. Goldblum, A. W. Reddie, and J. C. Reinhardt, "War-games as experiments: The project on nuclear gaming's signal framework," Bulletin of the Atomic Scientists, 2019.

[3] C. Madeira, V. Corruble, G. Ramalho, and B. Ratitch, "Bootstrapping the learning process for the semi-automated design of challenging game ai," in Proceedings of the AAAI-04 Workshop on Challenges in Game Artificial Intelligence, 2004, pp. 72–76.

[4] Sutton, R, and Barto, A. Reinforcement Learning: An Introduction, MIT Press, 1998.

[5] Mnih V, Kavukcuoglu K, Silver D, et al., "Human-level control through deep reinforcement learning," Nature, 2015, 518(7540), pp. 529–533.

[6] Silver D, Schrittwieser J, Simonyan K, et al., "Mastering the game of go without human knowledge," Nature, 2017, 550(7676), pp. 354–359.

[7] Berner C, Brockman G, Chan B, et al., "Dota 2 with large scale deep reinforcement learning," arXiv preprint arXiv: 1912.06680, 2019.

[8] Vinyals O, Babuschkin I, Czarnecki W M, et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," Nature, 2019, 575(7782), pp. 350–354.

[9] Levine S, Finn C, Darrell T, et al., "End-to-end training of deep visuomotor policies," The Journal of Machine Learning Research, 2016, 17(1), pp. 1334–1373.

[10] Shalev-Shwartz S, Shammah S, Shashua A. "Safe multi-agent reinforcement learning for autonomous driving," arXiv preprint arXiv:1610.03295, 2016.

[11] Sutton, Richard Stuart, D. Precup, and S. P. Singh, "Between mdps and semi-mdps," Artificial Intelligence, 1999.

[12] Thomas G. Dietterich, "The MAXQ Method for Hierarchical Reinforcement Learning," in Proceedings of the Fifteenth International Conference on Machine Learning, 1998.

[13] Dietterich, Thomas G. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition," Journal of Artificial Intelligence Research, 1999.

[14] Parr, Ronald , and S. Russell, "Reinforcement Learning with Hierarchies of Machines," Advances in Neural Information Processing Systems 10, 1998.

[15] Ronald Edward Parr. "Hierarchical control and learning for markov decision processes," University of California, Berkeley, 1998.

[16] Vezhnevets, Alexander Sasha, et al., "FeUdal Networks for Hierarchical Reinforcement Learning," in Proceedings of the 34th International Conference on Machine Learning, 2017, pp. 3540–3549.

[17] P. Dayan, G. E. Hinton, "Feudal Reinforcement Learning," Advances in Neural Information Processing Systems 5, Conference, 1992.

[18] Haotian Fu, Hongyao Tang, et al., "Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces," in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, 2019, pp. 2329–2335.

[19] Rashid T, Samvelyan M, De Witt C S, et al., "Q-Mix: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning," Proceedings of the 35th International Conference on Machine Learning, 2018.

[20] Rusu A A , Colmenarejo S G , Gulcehre C , et al., "Policy Distillation," in 4th International Conference on Learning Representations, 2016.

[21] Sanmit N, "Curriculum Learning in Reinforcement Learning," in Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 2017.

[22] Ha D, Dai A, Le Q V. "Hyper Networks," in 5th International Conference on Learning Representations, 2017.