Appendix A

Training settings

It is worth noting that due to the simplicity of thought-game (TG) and the smoothness of the curriculum in TG, our RL algorithm in TG and real games did not use any hand-designed rewards, but instead used the sparse final result as rewards. That is, in the final step, the reward is 1 for the victory, -1 for the loss, and 0 for the draw.

The number of iterations of training is generally set to 800. Each iteration runs 100 full-length games of SC2. One full-length game of SC2 is defined as an episode. The maximum number of frames for each game is set to 18,000. In order to speed up learning, we have adopted a distributed training method. We used 10 workers. Each worker has 5 threads. Each worker is assigned several CPU cores and one GPU. Each worker collects data on its own and stores it in its own replay buffer. Suppose we need 100 episodes of data per iteration, then each worker's thread needs to collect data for 2 episodes. Each worker collects the data of 10 episodes and then calculates the gradients, then passes the gradients to the parameter server. After the parameters are updated on the parameter server, the new parameters are passed back to each worker. Since the algorithm we use is PPO, the last old parameters are maintained on each worker to calculate the gradients.

Multi-processes parameters are as follows: the number of processes is set to 10, the number of threads in each process is set to 5, max-iteration I is set to 500. It is noted that update-num M is set to 500 on TG and 100 on the real game (RG) environment because simulation speed in thought-game is much faster than the original environment.

Network structure

It is worth noting that our network is a flat and simple structure compared to the previous method [1–3]. The network includes the policy net and value net. Both nets accept the current state as input. The policy net is a neural network with two hidden layers, with 64 neurons in each hidden layer. The policy net output action has the same dimension as the dimension of the macro-action. The value net is similar to the policy net. It is also a neural network with two hidden layers, each of which has 64 neurons. Value net outputs a prediction of the V value, so its dimension is 1. We have tried to change the number of layers of the neural network and the neurons in each layer and found that there is not much impact. The current default settings can train a policy very well.

Hyper-parameters of RL

PPO related parameters are as follows. The clip-value in the PPO algorithm is 0.2. The total loss calculation formula is as follows:

$$L_{total} = L_{clip} + c_1 * L_{vf} - c_2 * L_{entropy}. \tag{1}$$

Among them, L_{clip} is the loss of the clip, L_{clip} is the value network loss, and $L_{entropy}$ is the entropy loss. $C_1 = 0.01, c_2 = 10^{-3}$. $\gamma = 0.9995,$.

The PPO training has a batch-size of 256. Epoch-num is set to 10. The initial learning rate is set to 10^{-4}. Other parameters can be seen in the open-source codes.

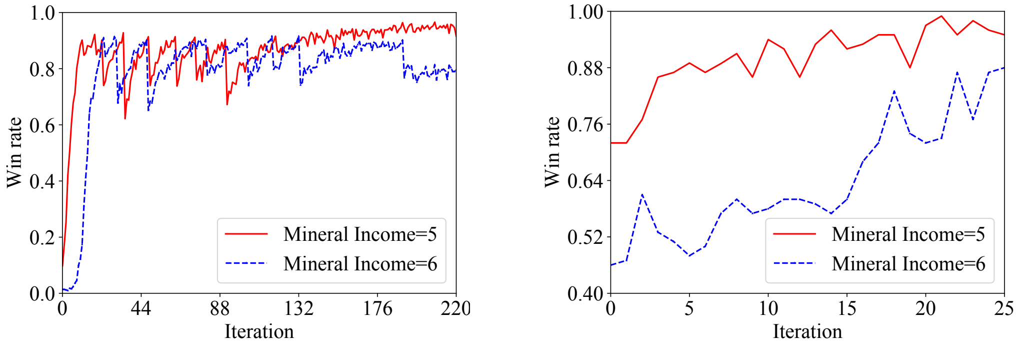
Appendix B

Impact of parameters in TG

We modify the two most important parameters affecting combat and economy. As can be seen from Fig.1 (a) and Fig.1 (b), the impact of economy parameters is small. We can still effectively train an agent in the TG and RG.

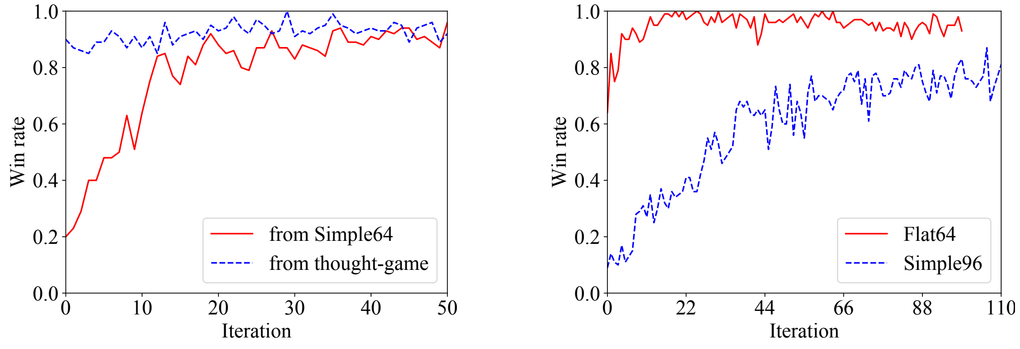
Extend to other maps

To test the scalability of our approach, we also trained our agent on other maps.



(a) (b)

Figure 1: (a) The effects of changing economy parameter (mineral income per one worker and one step) in thought-game. (b) The effects of changing economy parameter in real game.



(a) (b)

Figure 2: (a) Using different initial policy transfer to AbyssalReef. (b) Using same initial policy transfer to different maps.

As can be seen in Fig.2 (a) when migrating to a new map, the agent using our method can still learn and eventually rise to nearly 100% winning percentage. It is noted that if we use the *Simple64* model, agents can learn faster. However, training on the model learned directly from TG is also possible, and the final win-rate is close.

The results of training on other two maps, namely *Flat64* and *Simple96* is shown in Fig.2 (b). All training starts with a pre-trained policy of the TG. It is worth noting that learning is difficult because of the large size of *Simple96*. However, it can be seen that the win-rate of our agent can still rise steadily.

The results of the detailed map test are in the Table 1.

Test	A Race	O Race	Map	1	2	3	4	5	6	7
Map test 1	Protoss	Terran	AR	100%	100%	100%	98%	97%	96%	99%
Map test 2	Protoss	Terran	F64	100%	100%	99%	99%	98%	98%	97%

Table 1: Extend map test of our method. Each evaluation runs with 100 games. A=Agent, O=Opponent, AR=*AbyssalReef*, S64=*Simple64*.

Player	A Race	H Race	Map	Result (agent:human)
SC1 player	Protoss	Terran	S64	5:0
SC2 novice	Protoss	Terran	S64	5:0
SC2 golden	Protoss	Terran	S64	4:1

Table 2: Play against human. Human players are restricted to not choosing blocking tactics because the agent did not see any opponents using blocking tactics at training time. A=Agent, H=Human, S64=*Simple64*.

Play against Human

In order to test the performance of our method in the battle against humans, we test our agent against three human players. The three human players are an SC1 player, an SC2 novice and an SC2 *Golden* level player. Human players are restricted to not choosing blocking tactics because the agent did not see any opponents using blocking tactics at training time. However, human players can use a series of micro-operations while our agent has no micro-operations module. The results are in Table.2.

The result of our agent's match with the *Golden* level player is 4:1. The lost game is due to that the human player continues to learn during the battle and found the weakness of our agent while our agent did not do any learning. Our agent's APM (action per minute) is only 37 while the player's APM is 47, which indicates the future growth potential of our agent.

Other results

Our method can efficiently train an effective agent, so we tested the results of training two other races, namely *Zerg* and *Terran*. The training process can be seen in 2. Economic and combat settings of *Zerg* are similar to those of the *Protoss* above. The difference is that when *Zerg* builds a unit, it needs an extra resource – Larva. In thought-game, the number of Larvae is set to increase by one for every two steps. There will be an additional Larva for every two steps if any queen exists. In the original game, the number of Larvae is controlled by the game, i.e., the Hatchery will generate a Larva every 11 seconds. The Queen with Spawn Larva spell can inject 3 Larva eggs into the Hatchery and then the targeted Hatchery will generate 3 additional Larvae 29 seconds later. We set this spell to be automatically cast before each policy step ends. In addition, the *Zerg* Drones will disappear after building structures, which is also reflected in the thought-game.

Our training time is almost $1/100$ of [2]. The detailed comparison of time can be seen in Table 3.

Method	t_{μ_1}	t_{μ_2}	t_{μ_3}	Training time	(in hours)
[2]	$3d3h11m36s$	$9h59m21s$	$9h18m29s$	$94h29m$	94.50
Ours	**20m15s**	**21m50s**	**23m17s**	**65m**	**1.08**

Table 3: Detailed comparison of training time. For [2], t_{μ_1} refer to training time in difficulty level-2 (L2), t_{μ_2} refer to L5, and t_{μ_3} refer to L7. For ours, t_{μ_1} refers to L1 in thought-game. t_{μ_2} refers to L2 to L7 in thought-game. t_{μ_3} refer to L7 in real game. h=hour, m=minute, s=second.

Appendix C

Analysis of performance and time

Here we analyze why our agents get better performance and efficiency. Why a better performance is gotten based on the TG? For example, although their starting points may be different, the learning algorithms they use are the same. We argue that this reason may be due to the characteristics of machine learning, especially deep neural networks (we use it as the structure of the policy). In deep neural networks, even with the same learning algorithm such as gradient descent, a better initialization will result in a different final effect. More importantly, even with the same initial win rate, the effect based on TG learning is still better than the baseline. This confirms our assumption that in the TG, the agent actually learned the understanding of the rules in the RG through curriculum learning, which promoted the learning better. From the perspective of network optimization, the parameters learned from TG allow the network to be initialized in a better position, and this position is not locally optimal or close to it.

The consumption of training time consists of several parts: 1. The time t_ω sampling in the environment, which depends on the simulation speed of the environment. 2. The required sample size m_μ of the training algorithm. Due to the characteristics of the model-free reinforcement learning itself, a large number of samples are needed to learn a better policy. In addition, the total sample size of the training is the sum of the sample sizes on all tasks in the context of curriculum learning. 3. The time cost of the optimization algorithm, t_η. Therefore, the total time overhead is $T_1 = (t_\omega + t_\eta) \cdot m_\mu$. The general sampling time is much longer than the gradient descent algorithm, *i.e.*, $t_\omega \gg t_\eta$, hence $T_1 \approx t_\omega \cdot m_\mu$.

Suppose our training process divides the task into k steps. Step k is our target task, and the previous step $k-1$ is the pre-training process for curriculum learning. Therefore, we can denote our training time as

$$t_\omega \cdot m_\mu = t_\omega \cdot (m_{\mu_1} + m_{\mu_2} + \cdots + m_{\mu_k}). \quad (2)$$

In our approach, we moved the part of the curriculum to the TG, in which the time cost for simulating is much smaller than in the RG. Assume that the time $t_\xi = 1/M \cdot t_\omega$ in the thought-game, and the amount of sample required by the last step of the task m_{μ_k} is $1/K$ of the total m_μ. Then the time T_2 required by our algorithm is $(1/M + 1/K) \cdot T_1$ (detailed derivation can be seen in the appendix). It shows that the speed-up ratio of the new algorithm is determined by the smaller value of the acceleration ratio M and the last task sample ratio K. So speed boost comes from the increase of simulation speed (because of the simplicity of TG) and the migration of the curriculum tasks from RG to TG (by using ACRL).

Appendix D

Features of state space

The state space mainly includes two types, that is, RG features and TG features. The list of features of RG is shown in Table 4. The content of the TG features is shown in Table 5. These state features are all of the Protoss agent. It is worth noting that the features in TG are also basically present in RG. Therefore, when the agent learns in the RG, the feature in the RG is mapped to the feature in the TG. The one exception is that the time in TG and RG cannot match. The approach we take is not to map the time in the RG to the time in the TG, but to always set the time feature in the TG to zero. At the same time, we add a property called collected-mineral to the TG to approximate the advancement of time.

Macro-actions

The macro-actions of the TG and RG are shown in Table 6.

References

[1] P. Sun, X. Sun, L. Han, J. Xiong, Q. Wang, B. Li, Y. Zheng, J. Liu, Y. Liu, H. Liu, and T. Zhang, "TStarBots: Defeating the Cheating Level Builtin AI in StarCraft II in the full game," *arXiv:1809.07193*, 2018.

[2] Z. Pang, R. Liu, Z. Meng, Y. Zhang, Y. Yu, and T. Lu, "On reinforcement learning for full-length game of starcraft," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pp. 4691–4698, 2019.

[3] D. Lee, H. Tang, J. O. Zhang, H. Xu, T. Darrell, and P. Abbeel, "Modular Architecture for StarCraft II with Deep Reinforcement Learning," in *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, (Edmonton, Canada), pp. 187–193, November 2018.

Table 4: Real game state features

features	remarks
opponent.difficulty	from 1 to 10
observation.game-loop	game time in frames
observation.player-common.minerals	minerals
observation.player-common.vespene	gas
observation.score.score-details.spent-minerals	mineral cost
observation.score.score-details.spent-vespene	gas cost
player-common.food-cap	max population
player-common.food-used	used population
player-common.food-army	population of army
player-common.food-workers	population of workers(probes)
player-common.army-count	counts of army
player-common.food-army / max(player-common.food-workers, 1)	rate of army on workers
* num of probe, zealot, stalker, etc	multi-features
* num of pylon, assimilator, gateway, cyber, etc	multi-features
* cost of pylon, assimilator, gateway, cyber, etc	multi-features
* num of probe for mineral and the ideal num	multi-features
* num of probe for gas and the ideal num	multi-features
* num of the training probe, zealot, stalker	multi-features

Table 5: Thought-Game state features

features	remarks
time-seconds	
minerals	minerals
vespene	gas
spent-minerals	mineral cost
spent-vespene	gas cost
collected-mineral	mineral + mineral cost
collected-gas	gas + gas cost
player-common.food-cap	max population
player-common.food-used	used population
player-common.army-count	counts of army
* num of probe, zealot, stalker, etc	multi-features
* num of pylon, assimilator, gateway, cyber, etc	multi-features
* num of probe for gas and mineral	multi-features

Table 6: Thought-Game macro-actions

macro-actions	remarks
Build-probe	
Build-zealot	
Build-Stalker	
Build-pylon	
Build-gateway	
Build-Assimilator	
Build-CyberneticsCore	
Attack	position is selected by other rules
Retreat	position is selected by other rules
Do-nothing	