Appendix A

Training settings

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 thought-game (TG) and 100 on the real game (RG) environment because simulation speed in thought-game is much faster than the original environment.

Network structure

We used PPO as the RL algorithm. The network includes policy net and value net. Both nets accept the current state as input. 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. Value net is similar to 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 perform 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. Initial learning rate is set to 10^{-4}. Other parameters can be seen in the open source codes.

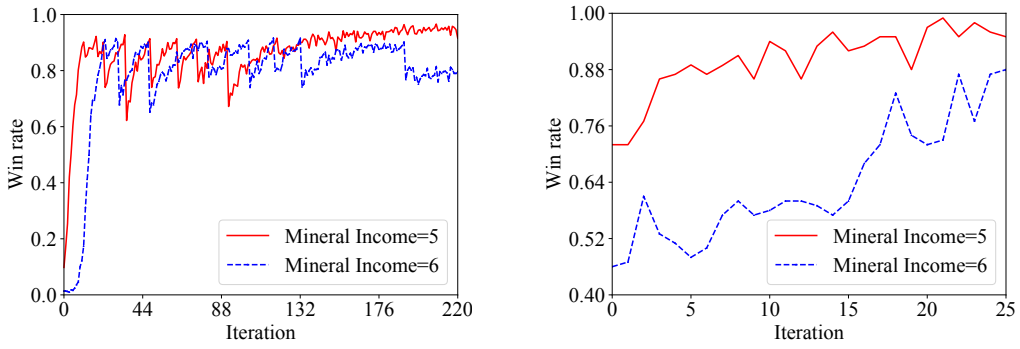
Appendix B

Impact of parameters in TG

We modify 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.

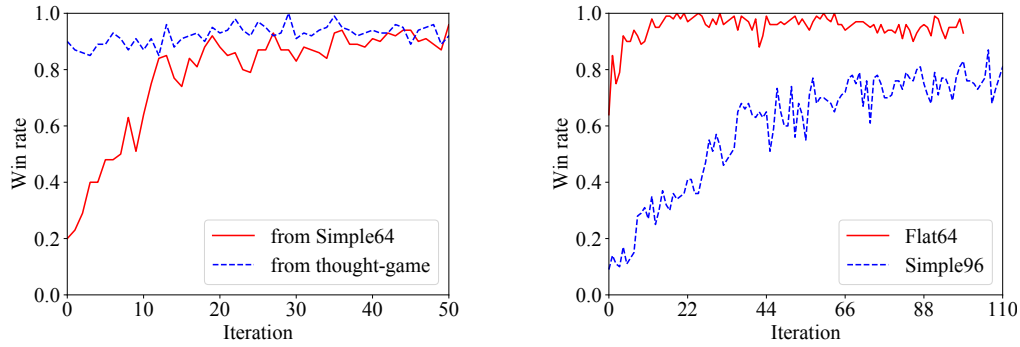
Test of other Races

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.



(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.

Test on another game

In order to verify that the TG idea is not just suitable for SC2, we tested it on another RST game which is "StarCraft: Brood War" (SC1). Although SC1 is the predecessor of SC2, there are many differences between the two. For example, their interfaces are inconsistent, and the details of each action are different. We operate the SC1 game based on TorchCraft [1] and use Tensorflow [2] for training. Although there are some differences in interface and units, SC1 and SC2 share many game rules and the victory condition, so we only need to make some adjustments to thought-game to use.

We use the *Protoss* race and train the agent against a *Terran* built-in bot on the map *Lost Temple*. It can be seen that the effect of direct training is very poor, and there is no obvious improvement for a long period of time. The transfer learning agent can not only have a better initial win rate, but also a fast increase. This may explain two points: First, thought-game does assist reinforcement learning in RTS games, whether on SC2 or SC1; Second, although the design of thought-game requires a certain amount of effort, it can bring a very large return (let some hard-to-train games become easier to train).

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 baseline. This conforms our assumption that in the TG, the agent actually learned the understanding of the rules in the RG through the 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

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

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

Table 1: 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 2: 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

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 1. The content of the TG features is shown on Table 2. 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 on Table 3.

Table 3: 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	

References

[1] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "Torchcraft: a library for machine learning research on real-time strategy games," *CoRR*, vol. abs/1611.00625, 2016.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pp. 265–283, 2016.