Measuring Generalization of Deep Reinforcement Learning Applied to Real-time Strategy Games

Shengyi Huang and **Santiago Ontañón** *
{sh3397,so367}@drexel.edu

Abstract

Reinforcement learning agents are often evaluated in the same scenarios as they are trained on, neglecting to evaluate the generalization abilities of the learned policies. However, in most real-world applications agents will not perform every time in the same exact environment, but need to be able to adapt to new situations. Although this topic has received increased attention recently, there is no existing work that studies this topic in the context of Real-time Strategy (RTS) games to the best of our knowledge. In this paper, we first introduce techniques to scale RL agents to play the full game with the primitive action space, and then evaluate the generalization ability of trained RL agents against other unseen opponents and maps.

Introduction

Real-time strategy (RTS) games pose a significant challenge for game Artificial Intelligence (AI) (Buro 2003; Ontanón et al. 2013). They are complex due to a variety of reasons: (1) players need to issue actions in real-time, which means agents have a very limited time to produce what is the next action to execute, (2) most RTS games are partially observable, i.e., a player might not always able to observe the opponents' strategies and actions, and (3) RTS games have very large action spaces. Moreover, recent application of Deep Reinforcement Learning (DRL) to RTS games introduces additional challenges such as (4) dealing with extremely sparse rewards (Vinyals et al. 2017, 2019) and (5) designing efficient observation and action space representations (Huang and Ontañón 2019). However, in the context of DRL, (6) *generalization* is arguably the biggest challenge. That is, given training against a given set of opponents and in a given set of maps, how well can the agents generalize to compete against an unseen set of opponents and maps?

This challenge is especially daunting for RL because the agents trained with RL are known to overfit the training environments (Gamrian and Goldberg 2019; Packer et al. 2018; Pinto et al. 2017). Prior work addresses this issue by creating a league of RL agents (the AlphaStar's League) to help diversify the training experience for the RL agents (Vinyals et al. 2019). However, it remains unclear exactly how well do RL agents generalize without such league training, which is rather computationally expensive. Studying the generalization of RL agents in RTS games could not only help us design techniques to create stronger agents, but also could help reduce the training cost by investigating more efficient training algorithms for generalization.

In this paper, our first contribution is to introduce important techniques to scale RL agents to play the full game of an RTS game simulator called μRTS. The first technique is to provide a complete invalid action mask that significantly improves the sample efficiency of the RL agents, while the second technique is to add an environment formulation that allows the RL agents to issue actions to all player owned units in the game. Then, we evaluate the generalization ability of trained RL agents against other unseen opponents and maps. Empirically, we find (1) primitive behaviors such as harvesting resources and producing many workers usually transfers to unseen maps while learned strategy rarely transfers, and (2) playing against a random opponent boostraps a more generalizable policy than playing against a strong deterministic opponent in the same map.

To support further research in this field, we make our source code and trained models available at GitHub[1], as well as all the metrics, logs, and recorded videos available at Weights and Biases[2].

Related Work

In recent work, there are mainly 3 ways to modify the testing environment to evaluate the generalization ability of trained RL agents: (1) visual changes, (2) different dynamics, and (3) different scenarios. We will briefly summarize the work on modification below.

Visual Changes. RL has achieved great success in training agents directly from pixels in domains such as Atari (Mnih et al. 2013). However, the agents usually overfit to visual cues of the environment that are irrelevant to the underlying MDP's dynamics (Song et al. 2019). So a popular modification to the testing environment is to change the visual cues such as replacing the game background image (Tang, Nguyen, and Ha 2020; Gamrian and Goldberg

*Currently at Google

[1]https://github.com/vwxyzjn/rts-generalization

[2]https://wandb.ai/vwxyzjn/rts-generalization

2019) or changing the color theme of the game (Lee et al. 2019; Cobbe et al. 2019).

Different Dynamics. In the robotics domain, obtaining real-world training data with RL is prohibitively expensive; it is desirable to train agents in simulations and transfer policies to the real-world environments (Packer et al. 2018) with potentially different dynamics compared to simulations. Packer et al. (2018); Pinto et al. (2017) evaluate the trained RL agents in continuous control tasks with different dynamics such as having different push force magnitude, or varying mass or lengths for the control objects.

Different Scenarios. In many games, different levels or maps require drastically different strategy to solve them. So another potential modification to the testing environment is to use different scenarios such as different game levels (Juliani et al. 2019) or maps (Perez-Liebana et al. 2019). To avoid manually defining these levels, a popular direction is to leverage Procedural Content Generation (PCG) to generate them (Cobbe et al. 2019; Justesen et al. 2018).

In RTS games, there are three ways to modify the testing environment: (1) different maps, (2) different opponents, and (3) different game rules (similar to different dynamics), but in this paper we just focus on the first two. While our experiments with different maps is similar to aforementioned work done with different scenarios, the observation input to our RL agents are low-level representation of game states while aforementioned work uses pixel inputs. This setting will arguably help us concentrate more on evaluating strategy generalization instead of high-level representation generalization. Furthermore, our experiments with different opponents is another class of different scenarios that is more similar to the generalization challenges found in the settings of Multi-Agent Reinforcement Learning (Lanctot et al. 2017), posing an additional dimension for generalization evaluation.

Evaluation Environment

We use μRTS[3] as our testbed and an extended version of gym-microrts[4] (Huang and Ontañón 2020b) as the RL interface to μRTS. μRTS is a minimalistic RTS game maintaining the core features that make RTS games challenging from an AI point of view: simultaneous and durative actions, long-term planning and real-time decision making. Additionally, the environment can be configured to be deterministic or non-deterministic, and the usual fog-of-war is also supported. However, for the experiments in this paper, we configure μRTS to be deterministic and fully observable (which is the most common setting in which this simulator is used). We now present the technical details of environment formulation for our experiments.

- **Observation Space.** Given a map of size $h \times w$, the observation is a tensor of shape (h, w, n_f), where n_f is a number of feature planes that have binary values. The observation space used in this paper uses 27 feature planes as shown in Table 1. A feature plane can be thought of as a concatenation of multiple one-hot encoded features. As an example, if there is a worker with hit points equal to 1, not carrying any resources, owner being Player 1, and currently not executing any actions, then the one-hot encoding features will look like the following:

$$[0,1,0,0,0],[1,0,0,0,0],[1,0,0],$$
$$[0,0,0,0,1,0,0,0],[1,0,0,0,0,0]$$

The 27 values of each feature plane for the position in the map of such worker will thus be:

$$[0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0]$$

- **Action Space.** Given a map of size $h \times w$, the action is an 8-dimensional vector of discrete values as specified in Table 1. The first component of the action vector represents the unit in the map to issue actions to, the second is the action type, and the rest of components represent the different parameters different action types can take. Depending on which action type is selected, the game engine will use the corresponding parameters to execute the action. As an example, if the RL agent issues a move south action to the worker at $x = 3, y = 2$ in a 10x10 map, the action will be encoded in the following way:

$$[3 + 2 * 10, 1, 2, 0, 0, 0, 0, 0]$$

Table 1: Observation features and action components.

Observation Features	Planes	Description
Hit Points	5	0, 1, 2, 3, ≥ 4
Resources	5	0, 1, 2, 3, ≥ 4
Owner	3	player 1, -, player 2
Unit Types	8	-, resource, base, barrack,worker, light, heavy, ranged
Current Action	6	-, move, harvest, return, produce, attack
Action Components	**Range**	**Description**
Source Unit	$[0, h \times w - 1]$	the location of unit selected to perform an action
Action Type	$[0, 5]$	NOOP, move, harvest, return, produce, attack
Move Parameter	$[0, 3]$	north, east, south, west
Harvest Parameter	$[0, 3]$	north, east, south, west
Return Parameter	$[0, 3]$	north, east, south, west
Produce Direction Parameter	$[0, 3]$	north, east, south, west
Produce Type Parameter	$[0, 5]$	resource, base, barrack, worker, light, heavy, ranged
Attack Target Unit	$[0, h \times w - 1]$	the location of unit that will be attacked

Training RL Agents to Play the Full μRTS Game

In previous work, gym-microrts (Huang and Ontañón 2020b) comes with two simplifications to make the learning task simpler: 1) its RL interface only selects one unit at

[3] https://github.com/santiontanon/microrts

[4] https://github.com/vwxyzjn/gym-microrts

a given *game tick* to perform actions while μRTS actually allows agents to issue actions to all units they own simultaneously, and 2) agents only need to generate one action each 10 ticks (so, 9 ticks are skipped), since many actions take at least 10 game ticks to complete execution. However, these settings become limitations when we want to evaluate the RL agent against existing bots such as the NaiveMCTS bot (Ontanón 2017), who at each game tick can issue actions to all the units the bot owns. So if we were to put the trained RL agent to a fight with the existing bots, the RL agent would be at a disadvantage.

In this work, we thus extend gym-microrts to circumvent these two limitations. Both are critical to our effort in creating agents to compete with existing bots, and thus being able to measure generality of the learned policies. The two main extensions are described in the following two subsections.

Complete Invalid Action Masking. Invalid action masking has been a popular technique to help the RL agents learn in games with large combinatorial action space by filtering out invalid actions (Vinyals et al. 2017; Berner et al. 2019). For example, Huang and Ontañón (2020b) provide a mask on the source and target unit selection in gym-microrts, which this filters out a lot of invalid actions that select non-existent source and target units. In this work, we further introduce masks on the action types and action parameters as well, which further filters out invalid actions such as harvesting from empty cells. We find this complete mask to significantly improve the sample efficiency of agents on all baseline tasks examined by Huang and Ontañón (2020b). Due to this improvement, we find skipping 9 game ticks still helpful but no longer necessary; therefore we do not skip game ticks in all experiments in this paper.

Issuing Actions to All Units Simultaneously. Issuing actions to multiple units simultaneously in RTS games is a significant challenge, as the number of possible action combinations grows exponentially with the number of units in the game state. First, the action space will be of varying size, which is difficult to handle using popular RL algorithm that assume a fixed-size action space. Second, even if the number of units remained fixed, the number of possible action combinations would be too large. Huang and Ontañón (2020b) simplifies this problem by only allowing the RL agent to issue one actions to a single unit at each game tick. To handle this issue more comprehensively, we extend gym-microrts in the following way: the RL agent still only issues an action to one unit at a time, but if not all player-owned units are issued an action, our environment execute this action in a simulated frame and return its observation and reward. Figure 1 illustrates this mechanism with an example.

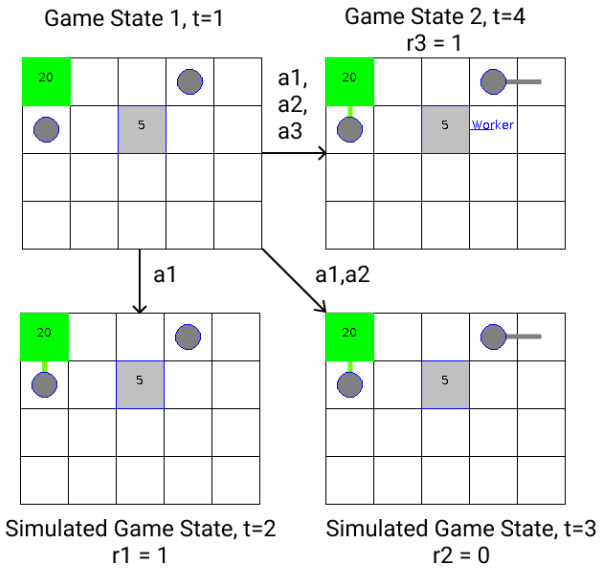


Figure 1: An example game state with 3 free units. The RL agent first issues an harvesting action to the bottom left worker; the environment returns a simulated game state that executed this action and a +1 reward for harvesting; it then issues an move action to the top right worker; the environment returns a simulated game state that executed the previous two actions and a 0 reward; it then issues a produce action to the base; the environment recognizes all the units are issued an action therefore return the real game state that executed the previous three actions. Its associated reward should be +2 for producing workers and harvesting, but since the harvesting reward has already been given out, we only return the +1 producing reward.

The Action Spaces of gym-microrts and PySC2

Although gym-microrts shares many similarities to PySC2 (the StarCraft II Learning environment (Vinyals et al. 2017)), it in general has a more fine-grained action space for the RL agents due to two reasons.

Gym-microrts has no AI-assisted actions. Consider the canonical task of harvesting resources and returning them to the base. In PySC2, the RL agent would need to issue two actions at two timesteps 1) select an area that has workers and 2) move the selected workers towards to a coordinate that has resources. Then, the workers will continue harvesting resources until otherwise instructed. Note that this sequence of actions is assisted by AI algorithms such as path-finding. After the workers harvest the resources, the engine also automatically determines the closest base for returning the resources, and repeating these actions to continuously harvest resources. So the challenge for the RL agent is to learn to select the correct area and move to the correct coordinates. In gym-microrts, however, the RL agent can only issue primitive actions to the workers such as "move north for one cell" or "harvest resource that is one cell away at north". Therefore it needs to constantly issue actions to control units at all times, having to learn how to perform these AI-assisted decisions from scratch.

Gym-microrts controls units individually. Per the action space formulation listed in Table 1 and our Unit Action Simulation module, the RL agent in gym-microrts issues actions to the all player-owned units at a given game tick. In comparison, an RL agent in PySC2 often issues actions for a *subset* of the player-owned units during the span of two or

Table 2: The list of hyperparameters and their values.

Parameter Names	Parameter Values
Total Time Steps	100,000,000
Number of Mini-batches	4
Number of Environments	8
Steps per Environment	128
γ (Discount Factor)	0.99
λ (for GAE)	0.95
ε (PPO's Clipping Coefficient)	0.1
η (Entropy Regularization Coefficient)	0.01
ω (Gradient Norm Threshold)	0.5
K (Number of PPO Update Iteration Per Epoch)	4
α (Learning Rate)	0.00025
Learning Rate Annealing	True
$c1$ (Value Function Coefficient)	0.5
$c2$ (Entropy Coefficient)	0.01

more game ticks. This is because PySC2 endeavors to create an action space similar to that of a human, so instead of issuing actions directly to the units the RL agent has to first "select" an area that has desired units at the current game tick, and then issue actions to the "selected" units at the next game tick. This is much like what a human would do in SC2. Although this makes it fairer to evaluate the RL agent against professional human players, the action space of gym-microrts is designed at a lower-level, where each unit can be controlled independently and simultaneously.

Training Algorithms

In all experiments conducted in this paper, we use PPO (Schulman et al. 2017) as the training algorithm. The hyper-parameters of our experiments can be found in Table 2. Our implementation is almost the same as that by Huang and Ontañón (2020a), with the exception that we also masks out invalid action types and parameters. For interested readers, their Appendix B provides more details on generating actions with multiple discrete values, performing invalid action masking, and utilizing code-level optimizations.

Experimental Setup

We train the RL agents against popular μRTS AIs and evaluate the RL agents on previously unseen maps and opponents. Specifically, we choose 4 AIs used in the 2020 μRTS competition. Three of them (RandomBiasedAI, WorkerRushAI, LightRushAI) are baselines built-in AIs, and the remaining one (CoacAI) is the winner of 2020 μRTS competition. Also, we use 1 map for training and 5 maps for evaluation.

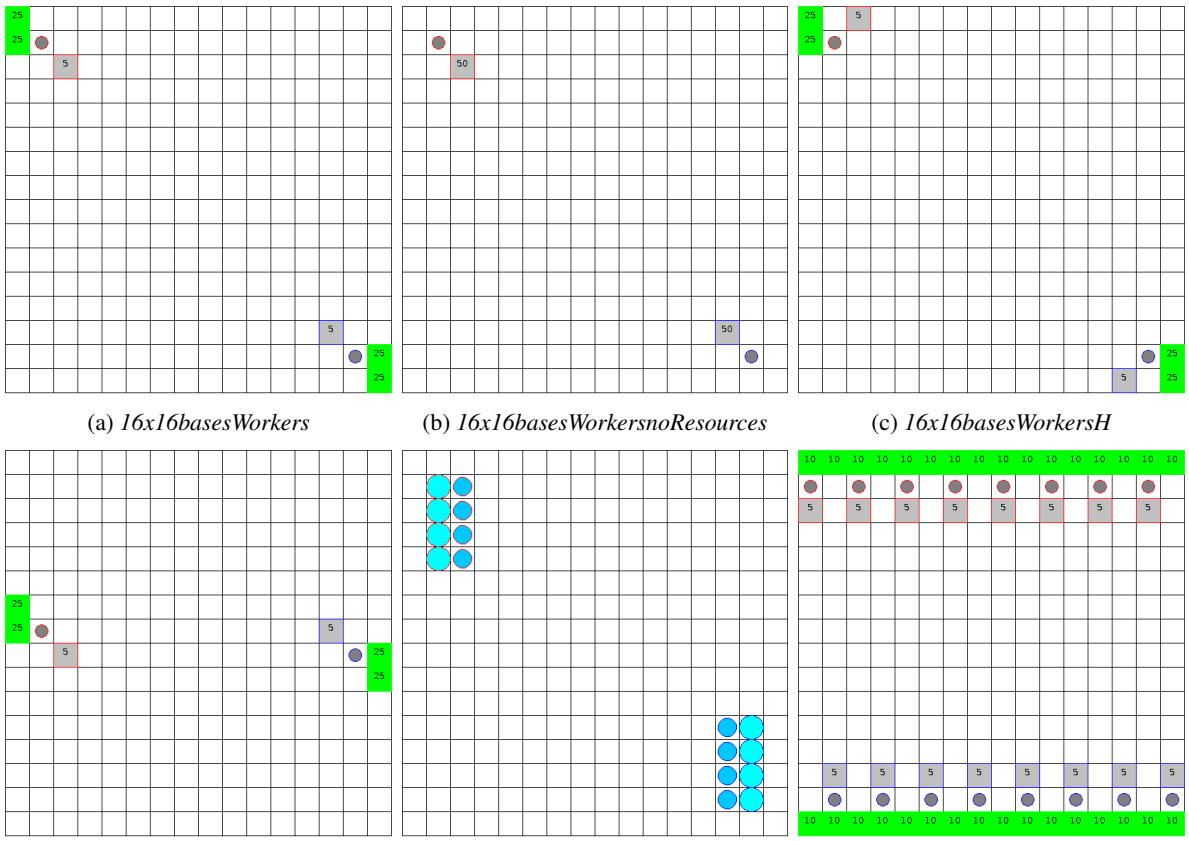
For our experiments, we provide a simple shaped reward function. The agent will get +10 for winning, +1 for harvesting one resource, +1 for producing one worker, +0.2 for constructing a building, +1 for each valid attack action it issues, +4 for each combat unit it produces. Here is a list of detailed description of the AIs and maps.

1. AIs:
   (a) RandomBiasedAI: This AI at large randomly executes available actions for units with a slightly higher chance to execute the harvest, return, and attack actions.
   (b) WorkerRushAI: This AI (1) uses a single worker to continuously harvest resources, (2) produces as many workers as it can, (3) sending the produced workers immediately to attack the nearby opponent's units and buildings.
   (c) LightRushAI: This AI (1) produces one barrack, (2) uses a single worker to continuously harvest resources, (3) produces as many light units as it can, (4) sending the produced light units immediately to attack the nearby opponent's units and buildings.
   (d) CoacAI: This is a hand-scripted AI, who won the 2020 μRTS competition against many other search and learning-based AIs. Its main strength is that it is very well fine-tuned to stop early rush strategies that are common among competition AIs. It features two phases: (1) Defense: all combat units stay close to the base for defending against an early rush, and (2) Attack: send all combat units to attack if in advantage.
2. Maps (As shown in Figure 2):
   (a) *16x16basesWorkers*: This is one of the standard maps used in the 2020 μRTS competition. It features one worker, one base, and two resource batches for both players. The map is large enough for more complicated strategies to be viable yet small enough to run fast experiments.
   (b) *16x16basesWorkersnoResources*: This is similar to *16x16basesWorkers* except with no resources to mine (but agents start with 50 resources already mined at game start).
   (c) *16x16basesWorkersH*: Similar to *16x16basesWorkers* except the bases position is a little off.
   (d) *16x16basesWorkersG*: Similar to *16x16basesWorkers* except the bases are postioned in the middle of the map. As a result, rushed-based strategies should be more successful.
   (e) *16x16meleeMixed8*: This is a combat only map where the players needs to position the combat units properly and target fire enemy units to win the battle.
   (f) *16x16eightBasesWorkers*: This is an extreme map with eight bases and workers with 16 resource batches. A lot of interesting strategies could be viable in this map.

For each of these AIs, we train RL agents with 4 random seeds in the standard *16x16basesWorkers* map for 100M time steps. For the maps listed above, the RL agents always start in the same position (top left) during training.

Experimental Results

After the training is finished, we use the ternary reward issued by the game engine itself (+1 for win, 0 for draw, and -1 for loss) to evaluate these RL agents against the other 3 previously unseen AIs and 5 previously unseen maps with



(a) *16x16basesWorkers* (b) *16x16basesWorkersnoResources* (c) *16x16basesWorkersH*

(d) *16x16basesWorkersG* (e) *16x16meleeMixed8* (f) *16x16eightBasesWorkers*

Figure 2: Maps used in the experiments reported in this paper.

10 random seeds. We report the results in Figure 3 averaged over 4 randomly seeded RL agents. The training curves showing both the shaped return as well as the ternary return (which the agents never see) are in Figure 4. Below are our analysis.

RL is viable in μRTS. After we augment the game with Unit Action Simulation and the agent with Complete Invalid Action Masking, a simple shaped reward function is sufficient to train the agents to perform competitively against all three baseline AIs (RandomBiasedAI, WorkerRushAI, LightRushAI). As shown in Figure 4a, the RL agents are able to optimize the shaped rewards well throughout the entirety of training, and as a consequence, as shown in Figure 4b, the agents are also get a growing win rate (i.e., they get a higher ternary return).

CoacAI remains a big challenge for our RL agents; after 100M time steps of training, the RL agents still lose most of the time, only getting around -0.7 ternary return. However, Figure 4b shows a rather promising curve for the RL agents trained against CoacAI, almost suggesting the agents would perform even better if we simply let the experiments run longer.

Although the rewards gives us plenty of information on the agents performance, it is of critical importance to watch what the agent is actually doing. After inspecting the videos of the agents playing the game, we find all agents are able to learn (1) harvesting resources almost optimally by using two workers to harvest resources continuously, (2) produce many workers in the beginning of the game, (3) generally move towards the enemy after having some units, and (4) engage the enemy in range. Behaviors (3) and (4) exhibit the most instability across agents: it is easy to spot instances where the agents is moving rather arbitrarily or not attacking properly. Here is a list of additional observations for each of our agents:

1. Agent trained against RandomBiasedAI: To our surprise, this agent actually learns quite a robust policy. It additionally learns to do a worker rush while building a barrack to transition into good mid-game play by start producing combat units.
2. Agent trained against LightRushAI: This agent generally (3 out of 4 random seeds) learns a behavior similar to the one trained against RandomBiasedAI. However the one with the best performing random seed learns to do a complete worker rush (which is very effective against LightRushAI), achieving almost a consistent +1 ternary return.
3. Agent trained against WorkerRushAI: This agent exhibits a wide range of behaviors for different random seeds.

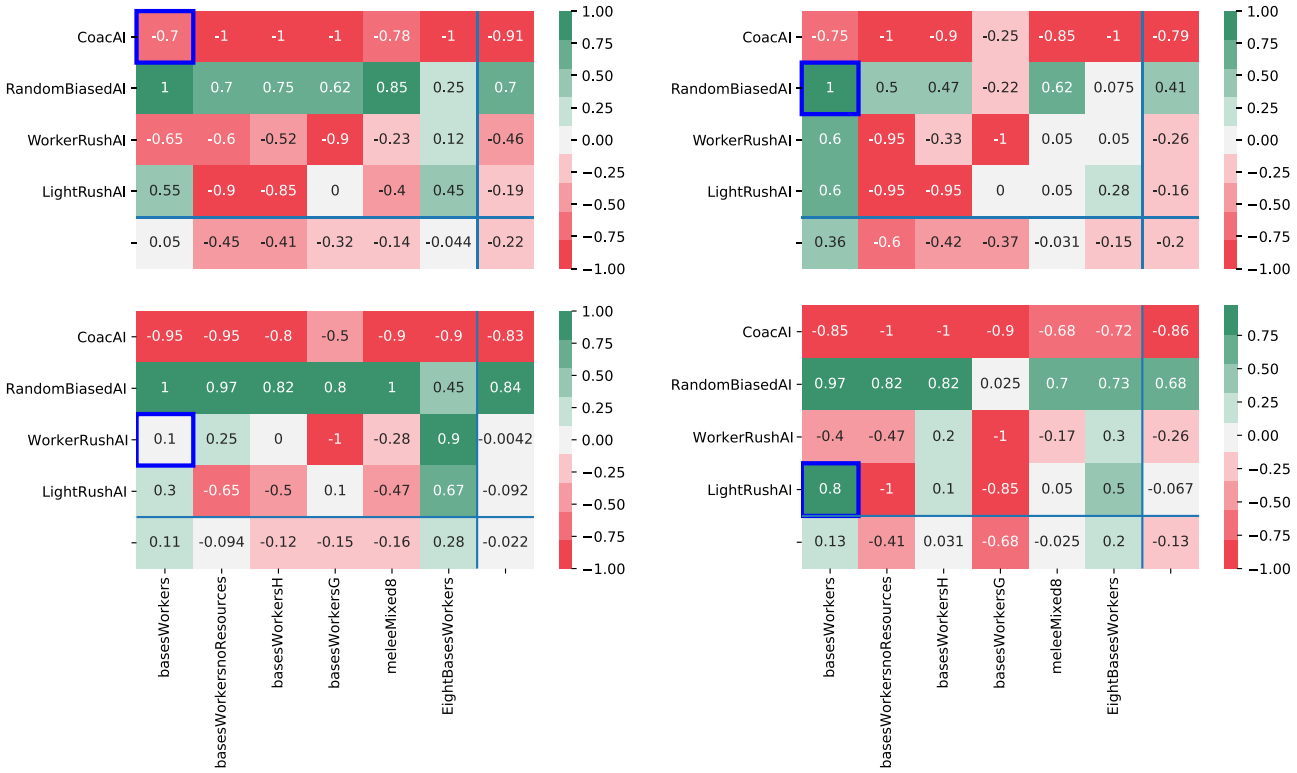


Figure 3: The blue-outlined cell contains the evaluation result for the RL agent using the training setups, and the other cells except the last row and column are evaluation results for unseen AIs and maps. Each cell shows the average ternary return (+1 for win, 0 for draw, and -1 for loss) for agents competing against the AI on the x-axis in the map on the y-axis. The last row and column are the column and row averages.

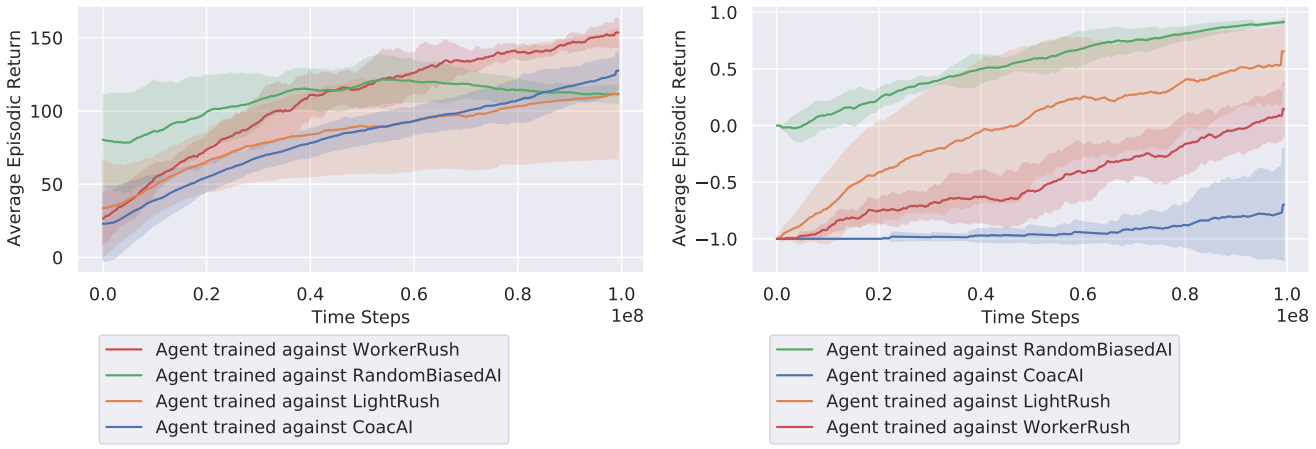
One seed learns to do a better worker rush than WorkerRushAI, one seed learns similar strategy as the agent trained against RandomBiasedAI, two seeds learn to do a rather strange and weak strategy that is between the strategies learned in previous two seeds.

4. Agent trained against CoacAI: The agent generally learns behavior similar to the one trained against RandomBiasedAI. It is sometimes able to defeat CoacAI with a perfect execution of worker rush. However, it usually loses the combat between combat units.

Convergence Time. Each agent of our experiments takes about 4 to 5 days to train by using the r5ad.large instance on AWS, which has a commodity machine configuration[5] (16GiB of RAM, 75 NVMe SSD, 2 vCPU). However, the trained agents are able to learn rather complex strategies in the full game from scratch. From a computational standpoint, gym-microrts does not demand high computational requirements to conduct research in RTS games. In comparison, conducting RL research with StarCraft II (Vinyals et al. 2019) and Dota II (Berner et al. 2019) requires significantly more computational resources.

[5]See https://aws.amazon.com/ec2/instance-types/ for the details of the configuration.

Opponent Generalization Results. Similar to the work done by Gao et al. (2019) in Pommerman, we found the use of random opponents helps with Opponent Generalization. The agent trained against RandomBiasedAI obtains a relatively good win rate against other three unseen AIs (-0.75 ternary return for against CoacAI and -0.6 ternary return for against WorkerRushAI and LightRushAI) in the same map that it was trained on, as shown in the first column of the tables in Figure 3. This is a rather surprising result for us because we had originally expected the agent trained against CoacAI, which is the strongest AI of our pool, to learn best strategy and generalize against other opponents. In reality, the agent trained against CoacAI actually performs quite poorly against other types of AIs, achieving the average ternary return of -0.65 (losing its battles against WorkerRushAI for most of time during evaluation). We suspect the performance behind the agent trained against RandomBiasedAI is due to a combination of two reasons: 1) the opponent RandomBiasedAI performs rather randomly and thus provides a much more diverse experience compare to other AIs that are mostly deterministic, and 2) the RL agents is simply optimizing against the shaped rewards and would learn a relatively good strategy regardlessly. So potentially having the shaped rewards and a random opponent are perhaps sufficient to obtain strong strategies in a given map.



(a) Episodic return in shaped reward

(b) Episodic return in ternary reward

Figure 4: The shaped return (left figure) and ternary return (right figure) over the 100M time steps of training for agents competing against selected AIs used in the 2020 μRTS competition.

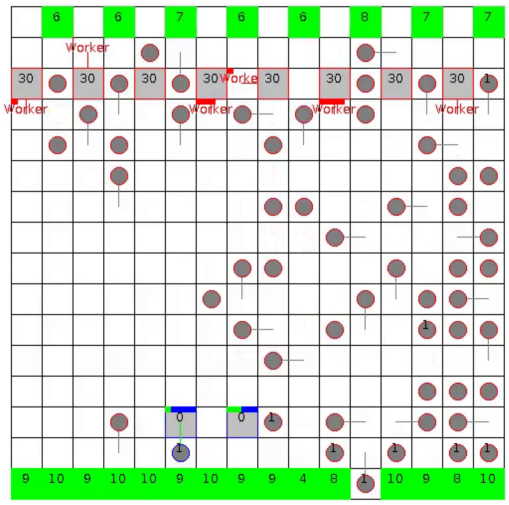


Figure 5: The screenshot shows that the RL agent trained against WorkerRushAI is able to carry out some useful behaviors such as harvesting resources, producing workers, and attacking enemy in the *16x16eightBasesWorkers* map that the RL agent has never seen before. Green squares are resources, grey circles are workers, and grey square are bases. See the complete video here: https://youtu.be/aJGUgGKXbh4

Maps Generalization Results. Upon visual inspection of the agents' behaviors in evaluation, we find the primitive behaviors such as harvesting and producing many workers could generalize to unseen maps. However, strategy generalization are much more problematic and we find mixed results. For example, the agent trained against WorkerRushAI successfully transfers strategy to even an extreme map *16x16eightBasesWorkers*. As shown in Figure 5, the agent almost carries over all useful behaviors learned in the original map to the new map to do a strong worker rush, thus resulting in a near-perfect ternary return of 0.9. However, the same agent failed poorly on another map *16x16basesWorkersG* with different starting position: it is able to harvest resources and produce workers immediately after the game has started, but soon after, it would perform rather erratically, sending workers to the strange locations or temporarily stop performing any actions without apparent purposes.

Overall Generalization Results. We find the agent trained against WorkerRushAI seems to obtain the best overall generalization result, achieving the best ternary return of -0.022 averaged over the entire table. Through visual inspection, we generally find the agent trained against WorkerRushAI is able to apply a strategy similar to worker rush to fight against unseen opponents and maps. An intuitive explanation behind this performance is that this simple strategy is easier to transfer to unseend maps and AIs compare to more sophisticated strategies learned by other agents.

Conclusions

In this paper, we evaluate the generalization ability of trained RL agents against other unseen opponents and maps in the context of RTS games. In order to do so, we present results on the μRTS game environment, training agents against a fixed opponent and in a fixed map, and then evaluating their performance against other opponents in unseen maps.

Empirically, we find (1) primitive behaviors such as harvesting resources and produce many workers usually transfers to unseen maps while learned strategy rarely transfers, and (2) playing against a random opponent boostraps more generalizable policy than playing against a strong deterministic opponent in the same map.

For future work, we would like to extend our the approach to train RL agents that can generalize to diverse opponents and map of arbitrary sizes.

References

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J. W.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *ArXiv* abs/1912.06680.

Buro, M. 2003. Real-time strategy games: A new AI research challenge. In *IJCAI*, volume 2003, 1534–1535.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2019. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588* .

Gamrian, S.; and Goldberg, Y. 2019. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *International Conference on Machine Learning*, 2063–2072.

Gao, C.; Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2019. Skynet: A top deep RL agent in the inaugural pommerman team competition. *arXiv preprint arXiv:1905.01360* .

Huang, S.; and Ontañón, S. 2020a. Action Guidance: Getting the Best of Sparse Rewards and Shaped Rewards for Real-time Strategy Games. *arXiv preprint arXiv:2010.03956* .

Huang, S.; and Ontañón, S. 2020b. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *arXiv preprint arXiv:2006.14171* .

Huang, S.; and Ontañón, S. 2019. Comparing Observation and Action Representations for Deep Reinforcement Learning in μRTS .

Juliani, A.; Khalifa, A.; Berges, V.-P.; Harper, J.; Teng, E.; Henry, H.; Crespi, A.; Togelius, J.; and Lange, D. 2019. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378* .

Justesen, N.; Torrado, R. R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729* .

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in neural information processing systems*, 4190–4203.

Lee, K.; Lee, K.; Shin, J.; and Lee, H. 2019. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv* arXiv–1910.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602* URL http://arxiv.org/abs/1312.5602.

Ontanón, S. 2017. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research* 58: 665–702.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in games* 5(4): 293–311.

Packer, C.; Gao, K.; Kos, J.; Krähenbühl, P.; Koltun, V.; and Song, D. 2018. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282* .

Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2019. General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games* 11(3): 195–214.

Pinto, L.; Davidson, J.; Sukthankar, R.; and Gupta, A. 2017. Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702* .

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Song, X.; Jiang, Y.; Du, Y.; and Neyshabur, B. 2019. Observational overfitting in reinforcement learning. *arXiv preprint arXiv:1912.02975* .

Tang, Y.; Nguyen, D.; and Ha, D. 2020. Neuroevolution of Self-Interpretable Agents. In *Proceedings of the Genetic and Evolutionary Computation Conference*. URL https://attentionagent.github.io. https://attentionagent.github.io.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782): 350–354.

Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A. S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; et al. 2017. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* .