Exploring the Impact of Reward Shaping on MicroRTS AI Agents

Timo Loher
tloher@student.ethz.ch

Samuel Räber
sraeber@student.ethz.ch

Abstract

Training reinforcement learning agents on tasks with sparse rewards is inherently difficult due to the very low frequency of feedback signals for the agents. This problem has been mitigated in the past by using shaped rewards. However, this solution has some drawbacks on its own. One major problem is that the agent learns to optimize different goal that often does not align well with the true objective. Action Guidance is a technique to use shaped rewards while still optimizing for the true rewards. In our work, we implemented Action Guidance and tested multiple different setups on the real-time strategy game MicroRTS to assess the impact of Action Guidance on the learning of an agent.

1 Introduction

Reinforcement learning agents have managed to surpass human experts in games like chess, shogi or Go[1] and video games like Atari 2600 games[2], StarCraft II[3] or Dota 2[4]. One of the problems faced when training such an agent is the sparsity of the received rewards in these games which often makes training hard[5]. Sparse rewards can prevent the agent from reaching any rewarding state which leads to a lot of exploration in a very large state and action space which then leads to sample inefficient training. Reward shaping[6] is a an approach to improve sample efficiency by providing additional, intermediate rewards which can steer the agent towards the final objective. This however leads to the problem that the agent no longer optimizes the true objective.
Action Guidance by Huang et al.[7] is a technique to use shaped rewards as a means to increase sample efficiency while still optimizing the true sparse reward. This is achieved with auxiliary agents that learn some easier objectives alongside the main agent and provide Action Guidance, meaning they override the main agents actions to steer it towards better results. We implemented the algorithm on a PPO agent for the game MicroRTS[8]. We extended the implementation to feature multiple auxiliary agents and performed tests on two different tasks inside the MicroRTS framework.

2 Related Work

2.1 Imitation Learning

On tasks with sparse or delayed reward signals imitation learning has been employed successfully in the past. There are a couple of different approaches:
Behavioral Cloning[9] formulates the problem as a supervised learning problem where the agent aims to learn a policy that matches the observed expert actions best. Subsequent works built on this fundamental idea. For example, *DAgger*[10] aggregates a dataset of expert demonstrations based on the current agents actions to improve sample efficiency.
Inverse RL[11] tries to recover the underlying reward function based on the observed expert trajectories.
A big drawback of imitation learning is the need for an expert to provide demonstrations. Thus it can

not be applied to problems that have not been solved yet. For example there are currently no human experts on MicroRTS as far as we know. Also the agent can at most get as good as the expert which might not be perfect to begin with.
Action Guidance can be seen as a type of imitation learning but instead of an expert we have auxiliary agents that are trained simultaneously.

2.2 Transfer Learning

Transfer learning[12] is a technique to learn a difficult task by learning an easier task first. The knowledge of the easier task is then transferred to the harder task which should make it easier to learn. This is similar to action guidance where auxiliary agents learn an easier task than the main agent, but compared to transfer learning where the same agent sequentially trains on multiple different tasks in action guidance multiple agents learn different tasks at the same time.

2.3 Hierarchical Reinforcement Learning

In Hierarchical Reinforcement Learning (HRL)[13] we try to break down the task we initially want to solve into smaller, easier tasks. Usually this involves a main agent that learns to solve the full task, as well as auxiliary agents that learn simpler tasks. We want to point out here that this is different than Action Guidance as in HRL the main agent uses the auxiliary agents after training to take actions, while in Action Guidance we only sample auxiliary agents during training to increase sample efficiency but after training the main agent has to act on its own.

3 Action Guidance

3.1 Overview

Huang et al.[7] propose a technique called Action Guidance: The idea is that we train auxiliary agents on shaped rewards at the same time as a main agent which is trained with the true sparse rewards. The auxiliary agents provide action guidance: During training, the main agent can take an action from one of the auxiliary agents instead of from his own policy. The probability of sampling an auxiliary agent starts out very high in the beginning (95%) and slowly decreases during training down to zero. During training, all agents are updated with the same rollout.
The goal of these auxiliary agents is to increase the sample efficiency. Because they have denser rewards they are learning faster than the main agent. Although they do not optimize the true reward we hope that they still learn useful behaviour that will be beneficial for the main agent to reach some rewards faster. Because the main agent only trains on the sparse rewards we can mitigate the problem of shaped rewards that the agent might optimize the wrong goal.
Even though Huang et al. formulated the algorithm with multiple auxiliary agents with differently shaped rewards there are no experiments performed to measure the impact of different auxiliary agent setups. That is why we performed experiments with four different auxiliary agent setups to compare the performance of each.

3.2 Implementation

The training with Action Guidance is split into three parts:

- Shift phase: At the start of the training the main agent receives action guidance with a fixed probability of 0.95
- Adaptation phase: The probability of action guidance decreases linearly down to 0. The agent has to rely more and more on its own actions.
- End phase: Until the training is finished the agent trains exactly the same as vanilla PPO.

To choose which auxiliary agent we want to sample an action from we choose uniformly random. Only exception is the experiment with 5 auxiliary agents, where we used sample weights similar to the shaped reward weights [1,1,0.2,1,4]. We did this because some actions are more important than others so we sample the agents specializing in these important actions more than others (for example, if we sample actions from the agent that optimizes building barracks too often we end up with way too many barracks).

4 Experiments

4.1 MicroRTS

All our experiments were conducted on the game MicroRTS.

4.1.1 Overview

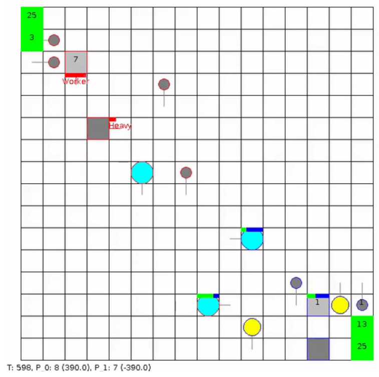
MicroRTS with its gym wrapper MicroRTS-py [8] is a RTS (real time strategy) game specifically developed for AI research. Compared to a traditional RTS game designed for human players there are a couple of advantages for AI:

- Simplicity: To make training RL agents as quickly as possible the game is kept at a minimum complexity. The game maps are kept very small and the actions that each unit can take is reduced to a minimum. The game has one type of resources, two types of building and four unit types. This is significantly simpler than commercial RTS games. Despite these simplifications, the core game play of RTS games and their challenges (gathering resources, micromanage units etc.) is still there.
- Interface: Usually, RTS games output an image and sound and take mouse clicks and keyboard keys as input. For an AI it is quite a challenging task to make sense of images and this is a field of research on its own. MicroRTS features a library for training AI agents called MicroRTS-py that removes this difficulty by giving the agents access to the game state directly. Agents can also issue inputs directly as opposed to a UI designed for humans.
- Performance: MicroRTS-py allows simulating games rather quickly due to the simplicity of the game itself and because it does not need to render an output image. This improves training times and allows for faster experiments.

Even though the game is very simplified there are some difficulties compared to other RTS games. There are no algorithmic aids whatsoever. In games meant to be played by humans unit can act quite smart. For example if you order a worker in Starcraft to collect resources it continues to do so on its own until otherwise instructed. In MicroRTS however the unit has to be moved manually to the resource, collect the resource and moved back to the base to drop it off.

4.1.2 Game Rules

MicroRTS is played on a grid of tiles. Each tile can hold either a unit, resources, a building or be empty. There exist 4 different units: worker, light unit, heavy unit and ranged unit. These units can move to adjacent tiles and attack enemy units and buildings. The worker is unique as it can create buildings and harvest resources. The two buildings are the base to build new workers and to drop off resources and the barracks to produce combat units. Stats of the different unit types and buildings can be seen in table below. The goal of the game is to destroy all enemy units and buildings by attacking them.



T: 598, P 0: 8 (390.0), P 1: 7 (-390.0)

Figure 1: A screenshot from a MicroRTS game on the same map used for our experiments.

MicroRTS Unit Statistics							
Unit Type	Cost	Health Points	Attack	Range	Move Time	Produced in	Production Time
Worker	1	1	1	1	10	Base	50
Light	2	4	2	1	8	Barracks	80
Heavy	2	4	4	1	12	Barracks	120
Ranged	2	1	1	3	10	Barracks	100

4.1.3 Interface

The observation that the agents recieve is of size $(h * w * 27)$ for a map of size $h * w$. For each tile, the agent recieves informations about the game in a one-hot encoded fashion. Details can be seen in the table below. Additionally the agent receives an *invalid action mask*. This reduces the possible amounts of actions significantly by masking out impossible actions. This is done by setting the logits of invalid actions before the softmax is applied to a very large negative value. The authors of MicroRTS-py showed in an ablation study that this dramatically improves performance. During a game of MicroRTS there are variable numbers of units needing actions and the units might have different numbers of available actions. However, RL algorithms usually require an action space with a fixed size. MicroRTS-py provides two different solutions. The first one is called *Unit Action Simulation* where the agent issues an action for each friendly unit sequentially. The second is called *GridNet* where the agent has to issue an action for every grid cell simultaneously. All actions for tiles where there are no units are then ignored by the game simulation. For our experiments we exclusively used GridNet, mostly because there is already a full implementation of PPO available for GridNet that saved us a lot of work.

Observation Features for a single Tile		
Features	Size	Values
Hit Points	5	0,1,2,3,≥ 4
Resources	5	0,1,2,3,≥ 4
Owner	3	player 1, neutral, player2
Unit Type	8	nothing, resource, base, barracks, worker, light, heavy, ranged
Current Action	6	nothing, move, harvest, return, produce, attack
Total	27	

Action Components for a single Tile		
Features	Size	Values
Action Type	6	nothing, move, harvest, return, produce, attack
Move Direction	4	up, right, down, left
Harvest Direction	4	up, right, down, left
Return Direction	4	up, right, down, left
Produce Direction	4	up, right, down, left
Production Type	7	resource, base, barracks, worker, light, heavy, ranged
Relative attack position	$(2r + 1)^2$ = 49	Relative target location, $r = 3$ is the maximal range of any unit
Total	78	

4.1.4 Rewards

There are six actions that yield a reward for the agents: Winning or losing the game (positive for winning, negative for losing), harvesting a resource, creating a worker, construct a building, issuing a valid attack and creating a combat unit. By weighting the rewards of these actions we can create different shaped reward functions. We will use a vector notation for the shaped reward weights in the same order as stated above. For example if we write rewards are $[10, 1, 0, 0, 0, 4]$ we mean the agent recieves +10 for winning, -10 for losing, +1 for harvesting a resource and +4 for creating a combat unit.

4.2 Experiment Tasks

We performed two different tasks within MicroRTS to test the performance of the agents.

4.2.1 Produce Combat Units (PCU)

In this task, the goal is to produce as many combat units as possible in 2000 game steps. The agent gets a reward of 1 for every combat unit created. This is a sparse reward, because the agent has to perform a series of non-trivial actions: Harvest resources, return resources to base, build a barrack and produce the combat unit. It is quite unlikely that this is achieved when taking random actions. Another difficulty is that in order to produce a combat unit there must be an empty tile adjacent to the barracks for the new unit to spawn, meaning that previously produced units must be moved away. For this task we used the 16x16 default map where the agent starts with a base and a worker. The opponent agent is set to passive, meaning it does not take any actions at all and does not interfere in any way. Both players have a total of 50 resources at their bases to collect. The cost of a barracks and combat units are five and two respectively. At game start the agent already has 5 resources. Assuming the agent does not gather resources from the enemy base (we never observed this behaviour) we can deduce a lower bound of the maximal score of 25.

4.2.2 Defeat Random Enemy (DRE)

In this task the goal is to win the game by defeating a simple enemy AI. Winning the game means killing all enemy units and buildings. Winning yields a reward of +1, losing -1 and drawing 0. A draw is concluded if none of the agents have won at the end of 2000 time steps. As the enemy we used RandomBiasedAI provided by MicroRTS. It chooses random actions with a bias towards moving units, which should be the most frequent action. The game is played on the same 16x16 standard map as in the PCU task described above. The best possible score is just 1 in this experiment.

4.3 Agent Setup

4.3.1 Architecture

The agent architecture has three parts: An encoder network, the policy network and the value network. The encoder network is a CNN that encodes the game state. It consists of four sequential convolutional layers. The policy network consists of four layers of transposed convolution. The value network is a standard feed forward neural network with one hidden layer. In total the agent has 838'767 learnable parameters. This agent is included in the MicroRTS-py library and we did not make any changes. Details about the architecture can be found in the appendix.

4.3.2 Action Guidance Setup

We ran several different setups for Action Guidance to test the effectiveness of different combinations of auxiliary agents. For all agents using action guidance we used a shift phase of 1'000'000 time steps and an adaptation phase of 3'000'000 time steps.

- **Baselines:** Vanilla PPO with shaped and sparse rewards. We used $[10, 1, 1, 0.2, 1, 4]$ as the reward weights for the shaped reward which are the default values.
- **1 Agent:** Action guidance with 1 auxiliary agent. Same weights as baseline $[10, 1, 1, 0.2, 1, 4]$. This is the same as originally proposed and tested by Huang et al.
- **2 Agents:** Action guidance with 2 auxiliary agents. As weights we used:
  - **A:**$[10, 1, 1, 0, 0, 0]$ for the first agent and $[10, 0, 0, 0.2, 1, 4]$ for the second agent.
  - **B:**$[10, 1, 1, 0, 0, 0]$ for the first agent and $[10, 0, 0, 0, 1, 0]$ for the second agent.

  Intuitively, the first agent should learn to extract resources while the second should learn to make units and attack in A or just to attack in B.
- **3 Agents:** Action guidance with 3 auxiliary agents. We split the second agent used in the 2 agent case into one learning to produce units $[10, 0, 0, 0.2, 0, 4]$ and one learning to attack $[10, 0, 0, 0, 1, 0]$. We left the agent to extract resources as it was in the 2 agents case. This is once again a split of task like a human player would discern different important sub tasks of an RTS game.
- **5 Agents:** Every agent optimizes a different reward type. Their weights are: $[0, 1, 0, 0, 0, 0]$,$[0, 0, 1, 0, 0, 0]$,$[0, 0, 0, 1, 0, 0]$,$[0, 0, 0, 0, 1, 0]$,$[0, 0, 0, 0, 0, 1]$. Additionally,

we did not uniformly sample the agents for action guidance but according to the weights of the original shaped reward (i.e. sample weights for agents are$[1, 1, 0.2, 1, 4]$)

- **Interpolated Rewards:** Vanilla PPO but we linearly interpolate between the shaped and sparse rewards. We start on the shaped rewards and end on the sparse reward at the end of the training. Formally, the agent receives the reward $r = t * [1, 0, 0, 0, 0, 0] + (1 - t)[10, 1, 1, 0.2, 1, 4]$, where t is the percentage of training time passed. We implemented this agent because it is a simpler method of achieving the two goals of action guidance: Efficient sampling with shaped rewards while still optimizing the true reward in the end.

4.4 Experiment Setups

For every experiment we trained for 5’000’000 total time steps. We performed a gradient update every 128 time steps with a batch size of 4, leading to a total of 4882 gradient updates. We did not tune any hyper parameters for PPO and left everything to default. For the action guidance experiments we started with a shift period of 1’000’000 steps, followed by 3’000’000 steps of adaptation and the rest without action guidance.
For the DRE task we ran the experiments multiple times with different seeds to measure and mitigate variance. This makes the results more reliable in general. Note that for the PCU task there is not much room to design multiple meaningful auxiliary agents. That is why we did not test multiple agents for that task and focused on DRE for multiple auxiliary agents. The exact code we ran can be found on GitHub.

4.5 Results

4.5.1 Produce Combat Units

Action guidance showed better results in the PCU experiment compared to the PPO baseline. After 5 million time steps the action guidance agent would produce about 10 combat units compared to 7.5 units produced by the PPO baseline. A human player can achieve at least 25 combat units relatively easy. This shows both agents are far from optimal for this task. This is largely due to the agents wasting resources on more workers or additional buildings (see Figure 2). But the action guidance agent does this to a lesser extent than the baseline, producing just one barracks in most of the rollouts and producing less workers. The baseline still increases the amount of buildings produced, since it is rewarded for buildings. The same behaviour can be seen with worker production, where the optimal behaviour would be to make no additional workers at all to use all available resources for combat units. Both agents produce more than 20 additional workers which indicates that they failed to learn to collect resources efficiently with few workers and instead collect resources by spamming workers to increase the chance of randomly collecting resources.

4.5.2 Defeat Random Enemy

We show the resulting sparse reward (win/loss) for DRE in a graph comparing all 7 agents in Figure 3. The shaped reward, interpolated reward and 1 auxiliary agent cases all learned to consistently achieve the maximal reward of 1 after 1 million time steps. The reward of the interpolating agent fell slightly towards the end when the reward got very sparse. The performance of action guidance with one auxiliary agent decreased more towards the end, only achieving the reward for winning a bit

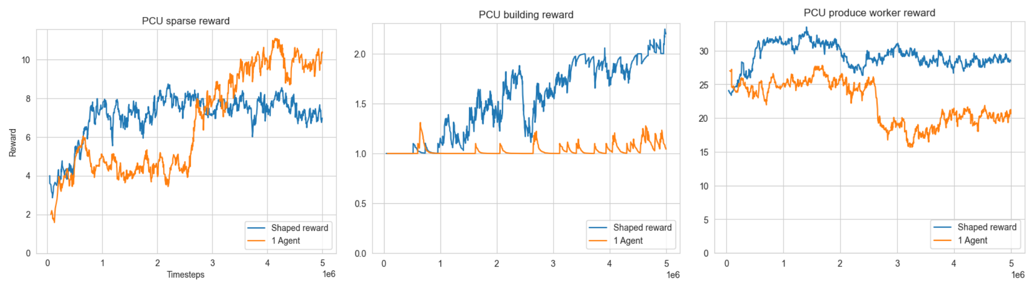


Figure 2: Sparse reward, reward for buildings and reward for producing workers for PCU experiments.

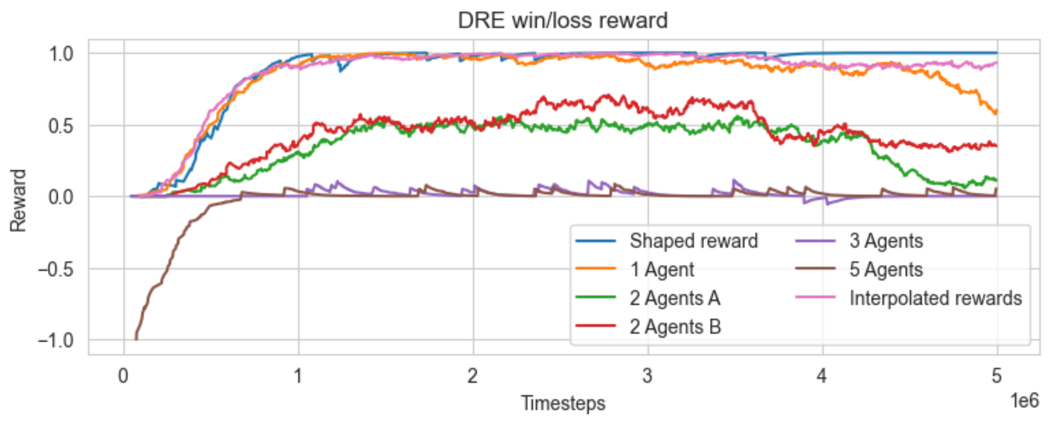


Figure 3: DRE sparse reward for the 7 agents used in our experiments. For 1 Agent, 2 Agents A and B and Interpolated rewards the reward is averaged over 5 runs of the experiment.

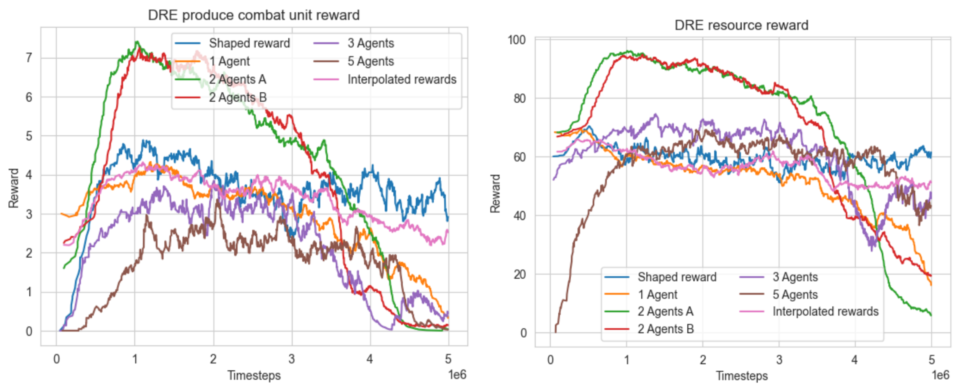


Figure 4: DRE combat units and resource harvest reward for the 7 agents used in our experiments. For 1 Agent, 2 Agents A and B and Interpolated rewards the reward is averaged over 5 runs of the experiment.

more than half the time. Action guidance with 2 auxiliary agents failed to ever learn a policy which consistently wins games, decreasing after the guidance stopped similar to the one agent case. With 3 or 5 auxiliary agents action guidance did not improve its winning chances, very rarely winning a game.

To further investigate the 4 agents which were in between a consistent reward of 1 or 0 we plotted the standard deviation of our experiments (see appendix). This shows that action guidance with one auxiliary agent is quite consistent among runs compared to the action guidance with two auxiliaries, which shows a larger variation. Especially agent B shows a huge difference between runs for the last million time steps, where there is no action guidance anymore.

The standard deviation of the interpolated rewards also tends to be a lot larger in the last portion of the run, where the reward approaches the sparse reward. This is mostly due to one of the five runs performing a lot worse in this portion than the other 4 (see appendix) which might be an outlier.

4.5.3 DRE analysis

To understand how the different agents play the game we analyzed how many resources were gathered and how many units were produced during the runs. this can be seen in Figure 4. While all agents manage to produce combat units, some action guidance agents produce a lot more than the PPO baseline and the interpolated rewards during the early stages of training. However they lose this behaviour towards the end of training when action guidance ends. The graph shows how this reward decreases steadily after the initial shift period, where maximal guidance is received.

A similar observation can be made for harvesting resources where some of the action guidance agents can be observed harvesting a lot less resources towards the end than at the 1 million time steps mark, showing the same steady decrease in reward. Action guidance using 3 and 5 auxiliary agents does not show the same behaviour, but the additional resources do not seem to help them win the game either.

Not producing combat units and harvesting less resources is not a problem in itself, since a *worker rush* (producing only a few workers and immediately attacking the enemy) is a valid strategy also employed by stronger bots. However since the win rate also drops we conclude that the agent did not learn this strategy.

These two observations lead us to the assumption that the time during action guidance (the shift and adaptation periods) were too short, so the main agent did not have enough time to learn from the behaviours of the auxiliary agents.

We observe that the agents generally learn to win games by flooding the map with units which increases the probability of a unit attacking the enemy by chance. The agents also have a very hard time to move units. After training even the best performing agents fail to move the units efficiently and goal oriented and the movement still looks random. That makes us believe that a shaped reward for unit movement (for example, distance to enemy units or base) would be very beneficial for all agents.

5 Conclusions and Future Work

Our experiments have shown that the idea of action guidance works with one auxiliary agent and even outperformed shaped rewards in a simple task. In the more complicated DRE task the PPO baseline performed best. Action guidance achieved worse results and did not learn the task quicker.

The main work of this paper, to perform experiments with multiple auxiliary agents giving action guidance, showed a decrease in performance compared to the PPO baseline. More agents led to slower training, achieved less rewards in a given amount of time steps and required more computing time per time step. Using too many agents even completely failed to learn the tasks.

This does however not mean action guidance can not work with multiple agents. Shaped rewards are difficult to tune and influence the performance of agents greatly. We chose the rewards for the agents by intuition. By tuning these rewards it might be possible to achieve way better performance than in our experiments.

We used a trivial sampling strategy for most agents, just sampling any auxiliary agent at random. It is very well possible one might achieve better results when using a different sampling strategy.

Exploring different sampling strategies is an idea which could be interesting for future research. It is also possible better results could be achieved using longer trainig times, with a longer adaption period and more training on just the sparse rewards afterwards.

Another interesting idea to explore is the true difference between shaped rewards which interpolate towards the sparse reward. Our experiment showed this to perform better for DRE than action guidance and it has the advantage of having to train just the one agent. If this idea can lead to better optimization towards the true sparse reward it might be interesting to explore and even be advantageous over action guidance.

References

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[3] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. 575(7782):350–354.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[5] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR, 06–11 Aug 2017.

[6] A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[7] Shengyi Huang and Santiago Ontañón. Action guidance: Getting the best of sparse rewards and shaped rewards for real-time strategy games, 2020.

[8] Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning, 2021.

[9] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[10] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[11] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery.

[12] Matthew Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 09 2007.

[13] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition, 1999.

A Appendix

A.1 Agent Architecture

Agent Architecture		
State Encoder		
Convolution Block (Conv2d + MaxPool + ReLU)	input: 16x16x27 output: 8x8x32	Kernel Size 3x3, Padding 1 Pooling Stride 2
Convolution Block (Conv2d + MaxPool + ReLU)	input: 8x8x32 output: 4x4x64	Kernel Size 3x3, Padding 1 Pooling Stride 2
Convolution Block (Conv2d + MaxPool + ReLU)	input: 4x4x64 output: 2x2x128	Kernel Size 3x3, Padding 1 Pooling Stride 2
Convolution Block (Conv2d + MaxPool + ReLU)	input: 2x2x128 output: 256	Kernel Size 3x3, Padding 1 Pooling Stride 2
Policy Network		
Transpose Convolution Block (ConvTranspose2d + ReLU)	input: 1x1x256 output: 2x2x128	Kernel Size 3x3 Padding 1, Stride 2
Transpose Convolution Block (ConvTranspose2d + ReLU)	input: 2x2x128 output: 4x4x64	Kernel Size 3x3 Padding 1, Stride 2
Transpose Convolution Block (ConvTranspose2d + ReLU)	input: 4x4x64 output: 8x8x32	Kernel Size 3x3 Padding 1, Stride 2
Transpose Convolution Block (ConvTranspose2d + ReLU)	input: 8x8x32 output: 16x16x78	Kernel Size 3x3 Padding 1, Stride 2
Value Network		
Fully Connected Layer	input: 256 output: 128	ReLU Activation
Fully Connected Layer	input: 128 output: 1	

A.2 Additional figures

The following figures show mean/standard deviation and the individual values for the experiments we ran multiple times.

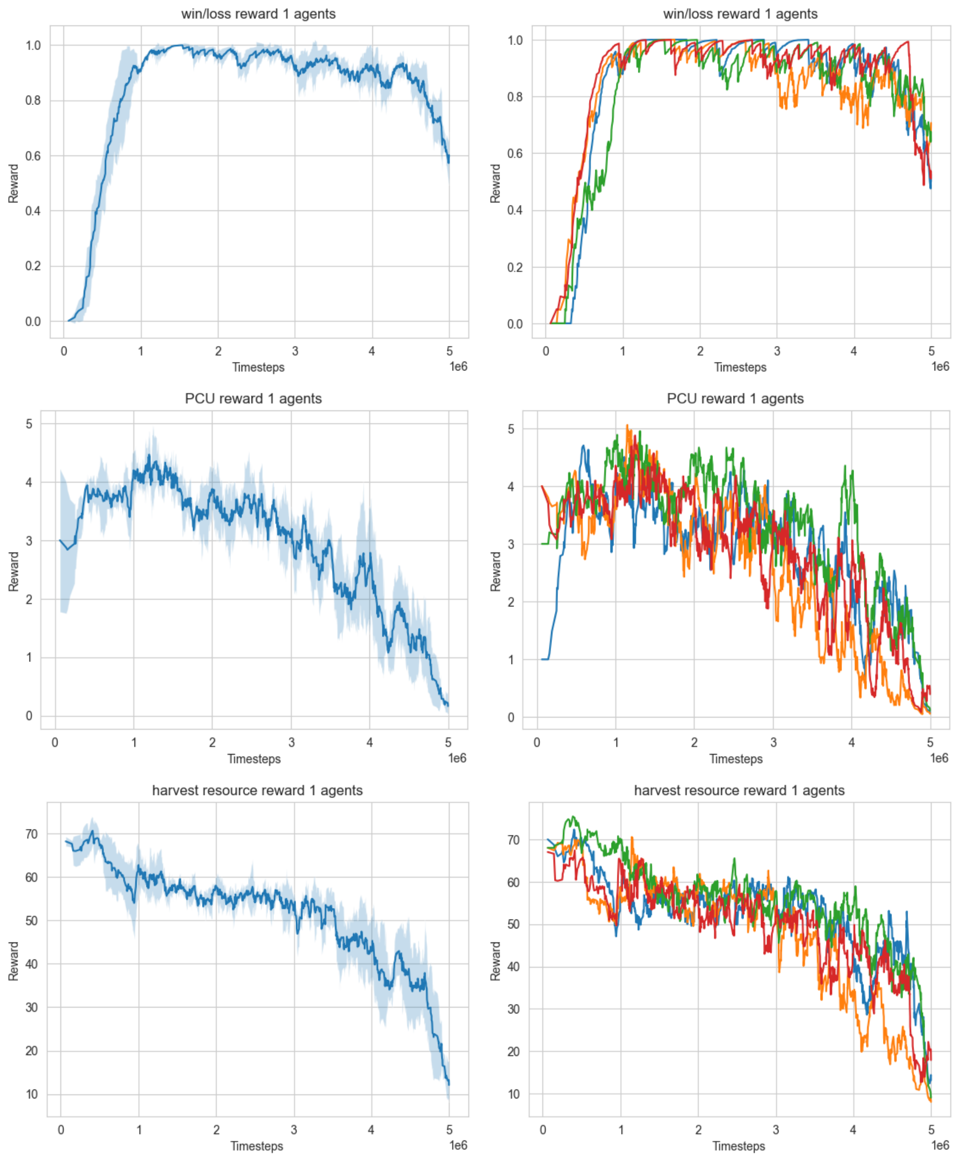


Figure 5: Individual plots for 1 Agent

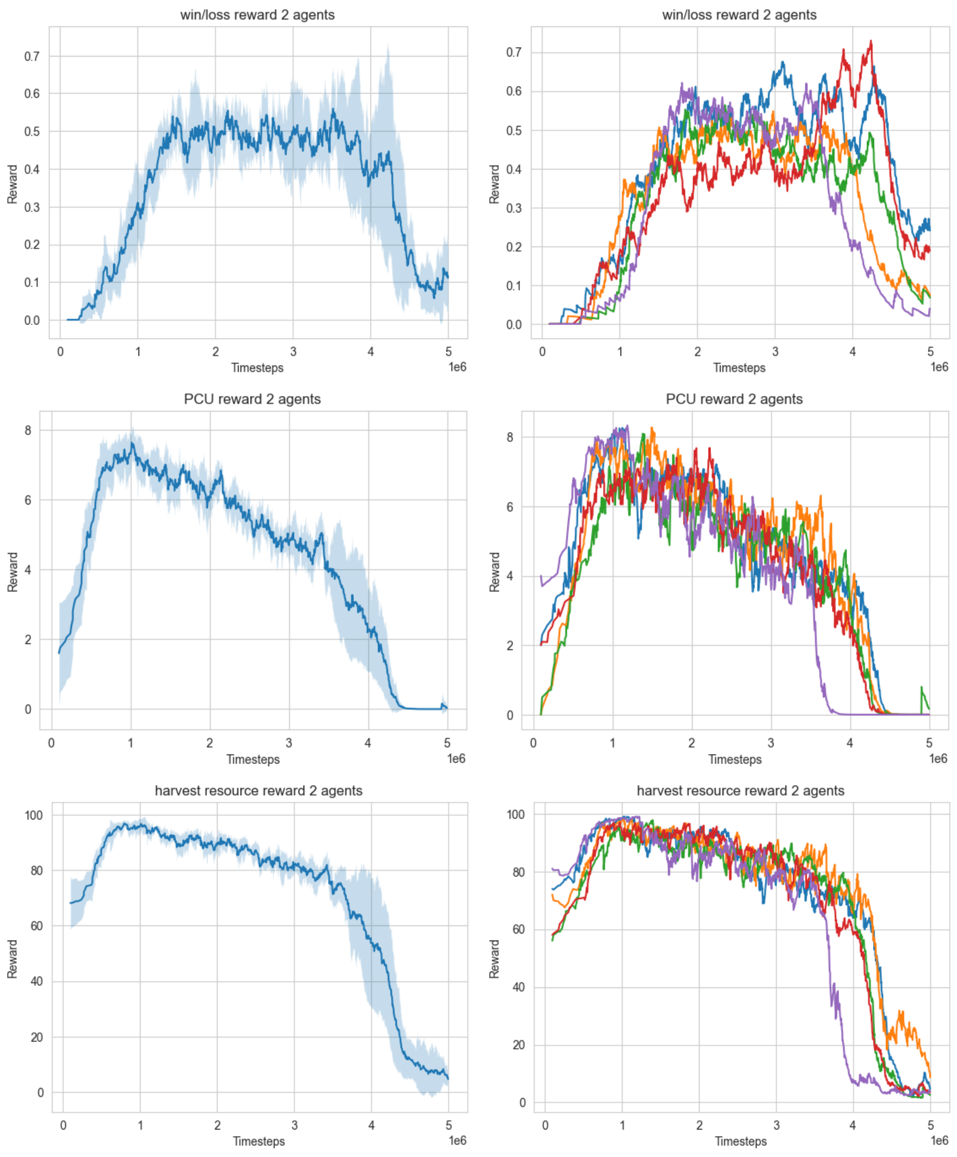


Figure 6: Individual plots for 2 Agents A

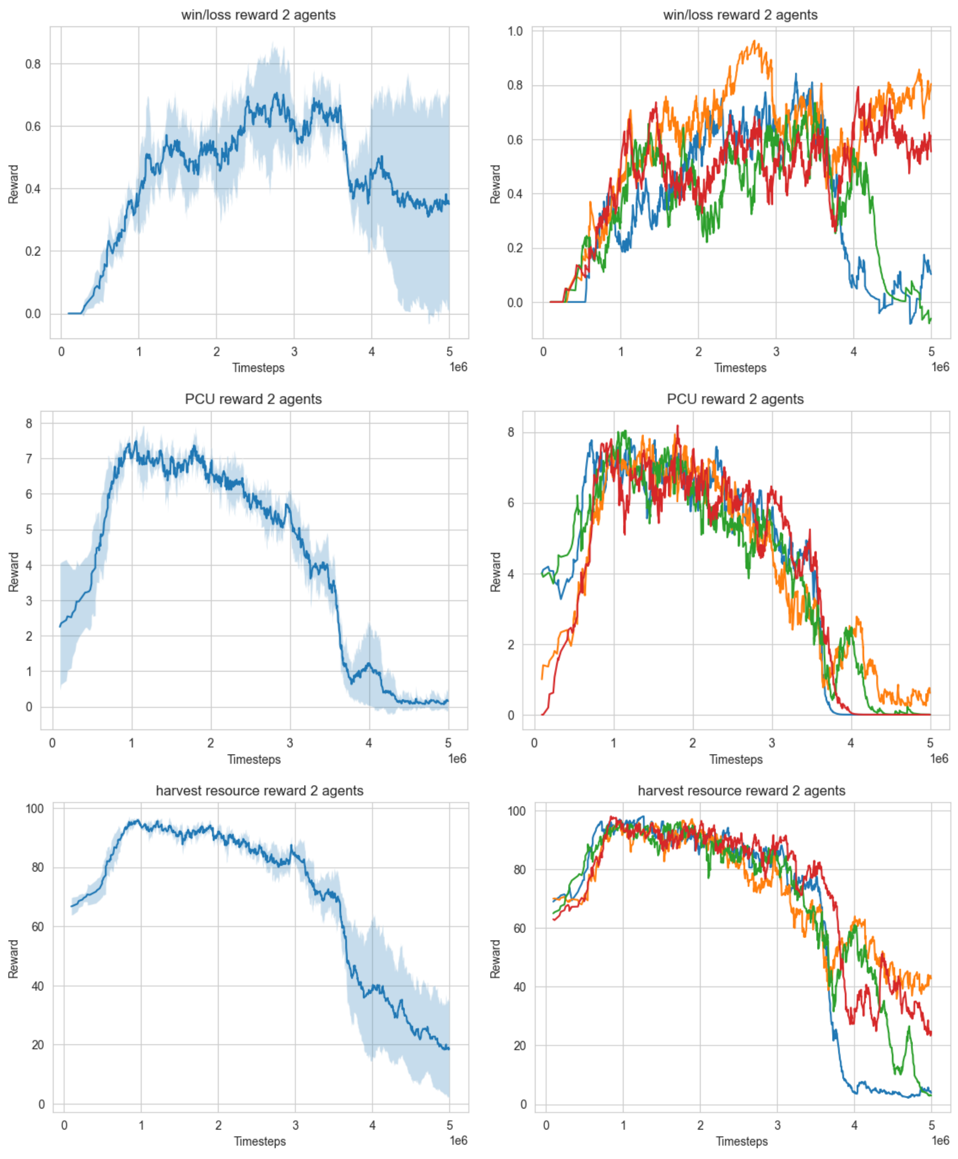


Figure 7: Individual plots for 2 Agents B

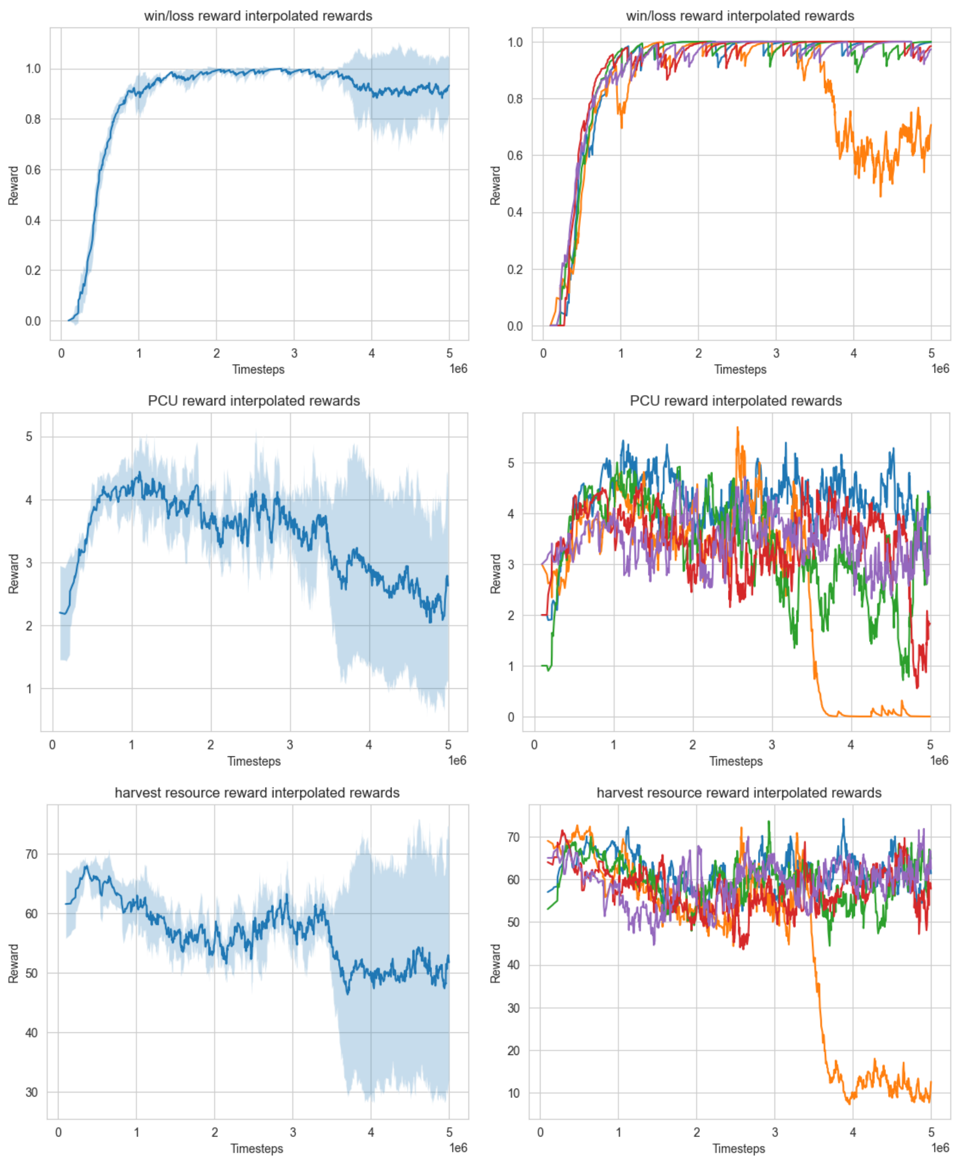


Figure 8: Individual plots for Interpolated rewards