PETTINGZOO: GYM FOR MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

OpenAI's Gym library contains a large, diverse set of environments that are useful benchmarks in reinforcement learning, under a single elegant Python API (with tools to develop new compliant environments). The introduction of this library has proven a watershed moment for the reinforcement learning community, because it created an accessible set of benchmark environments that everyone could use (including wrapper important existing libraries), and because a standardized API lets RL methods and environments from anywhere be trivially exchanged. This paper similarly introduces PettingZoo, a library of diverse sets of multi-agent environments under a single elegant Python API, with tools to easily make new compliant environments.

1 INTRODUCTION

Reinforcement Learning ("RL") considers learning a policy — a function that takes in an observation from an environment and emits an action — that achieves the maximum expected discounted reward when playing in an environment. OpenAI Gym (Brockman et al., 2016) was introduced shortly after the potential of reinforcement learning became widely known with Mnih et al. (2015). At the time, doing basic research in reinforcement learning was a large engineering challenge. The most popular set of environments were Atari games as part of the Arcade Learning Environment ("ALE") (Bellemare et al., 2013). The ALE originally was challenging to compile and install, and had an involved C API and later an unofficial fork with a Python wrapper (Goodrich, 2015). A scattering of other environments existed as independent projects, in various languages, all with unique APIs. This level of heterogeneity meant that reinforcement learning code had to be adapted to every environment (including bridging programming languages). Accordingly, standardized reinforcement learning implementations weren't possible, comparisons against a wide variety of environments were very difficult, and doing simple research in reinforcement learning was generally restricted to organizations with software engineering divisions. Gym was created to promote research in reinforcement learning by making comprehensive bench marking more accessible, by allowing algorithm reuse, and by letting average machine learning researchers access the environments. This last point was achieved by putting every environment that a researcher would want to benchmark with (at the time of creation) under one simple API that anyone could understand, in Python (which was just starting to be the *lingua-de-franca* for machine learning). This lead to a mass proliferation of reinforcement learning research (especially at smaller institutions), many environments compliant with the API (Kidziński et al., 2018; Leurent, 2018; Zamora et al., 2016), and many RL libraries based around the API (Hill et al., 2018; Liang et al., 2018; Kuhnle et al., 2017).

Multi-Agent Reinforcement Learning (MARL) in particular has been behind many of the most publicized achievements of modern machine learning — AlphaGo Zero (Silver et al., 2017), OpenAI Five (OpenAI, 2018), AlphaStar (Vinyals et al., 2019) — and has seen a boom in recent years. However, the field is in a similar state to reinforcement learning before the release of Gym. Popular benchmark environments are scattered across many different locations (or made from scratch), are based around heterogeneous APIs, and are often in unmaintained states. Because of this, highly influential research in the field is generally restricted to institutions with dedicated engineering teams, research into new methods generally aren't compared in like environments, and progress has been slow in comparison with single agent reinforcement learning (though this obviously cannot be attributed to benchmarks alone).

Motivated by this, we developed PettingZoo — a Python library collecting maintained versions of all popular MARL environments under a single simple Python API, that is very similar to that of Gym. It's on PyPI and can be installed via `pip install pettingzoo`.

2 Design Philosophy

Simplicity and Similarity to Gym

The ability for the Gym API to be near instantly understood has been a large driving factor in it's widespread adoption. While a multi-agent API will inherently add complexity, we wanted to create a similarly simple API, and one that would be instantly familiar to researchers who have worked with Gym.

Agent Environment Cycle Games Based API

Most environments have APIs that model agents as all stepping at once (Lowe et al., 2017; Zheng et al., 2017; Gupta et al., 2017; Liu et al., 2019; Liang et al., 2018), based on the Partially Observable Stochastic Games (POSGs) model. It turns out this easily results in bugs and is undesirable for handling strictly turn-based games, like chess, since every agent isn't allowed to step at once there. We instead model our API after the new Agent Environment Cycle games model (Terry et al., 2020b), which treats each agent as stepping sequentially. That is, an agent performs an action, the environment responds, the next agent acts, the environment responds again, and the cycle repeats. AEC has been shown to be equivalent to POSGs, which means the AEC paradigm can be used to model turn-based and parallel games. The paper introducing this model expounds on these benefits at great length.

Sufficient Configurability

We wanted to make environments that are highly configurable by arguments the norm. In Gym, environments are generally not configurable, and arguments at generation are not used at all. However, playing with various environment properties is often highly desirable, so this has been embraced by Gym environments outside the official library, as this makes research easier and aids reproducibility. Accordingly, we tried to make every reasonable environment parameter an option for users in PettingZoo.

This notion of configuration extends beyond environment configuration to how learning methods interact with the environment. Due to the wide diversity of optimizations and different strategies applied for MARL, we wanted our API to allow for low level access to rewards, observations, done states and other info, while still being very simple for normal applications. Cyclically expansive curriculum learning from Terry et al. (2020b) is a good example of an interesting method that requires this sort of low level access.

Quality of Life Improvements

Being users of Gym ourselves, we sought to add several "quality of life" improvements in PettingZoo motivated by frustrations we faced as users. These are:

- Comprehensive, production-grade continuous integration testing. Testing in Gym is arguably lacking, which has lead to issues in the past.
- Tests of environments for API compliance and proper functionality, both for end users and for continuous integration testing of the library. We also provide detailed recommendations for better practices, inspired by the well liked messages of the Rust compiler.
- Good error messages and warnings. When using Gym, triggering an error yields a trace back that needs to be slowly decoded to find the actual problem. We added speciality error messages and warnings for all common errors (that we're aware of) to make development and debugging easier. This is again inspired by the Rust compiler.
- Detailed, comprehensive documentation. Documentation is a fundamental part of a user-friendly software library. Observation space, action space, reward schemes, and other notable environment details are something you generally need to know to begin conducting even the most basic research with an environment. One criticism of Gym is that almost all information is only found in the source code, something especially problematic when working with sets of environments. To solve this in PettingZoo, we created a user friendly

wiki-styled website that clearly includes all relevant information for an environment, as well as general information for sets of environments. Our website also includes details about tests, comprehensive API documentation, and so on. This is discussed further in section 5.

3 API

Per our discussion above, we sought to create a simple API that could encapsulate all games and be instantly understood to any Gym user, illustrated by comparing Figure 1 and Figure 2.

Figure 1: Basic Usage of Gym

```
import gym
env = gym.make('CartPole-v0')
obs = env.reset()
for _ in range(1000):
    env.render()
    action = policy(obs)
    obs, reward, done, info = env.step(action)
env.close()
```

Figure 2: Basic Usage of PettingZoo

```
from pettingzoo.butterfly import pistonball_v0
env = pistonball_v0.env()
observation = env.reset()
for agent in env.agent_iter(1000):
    env.render()
    observation, reward, done, info = env.last()
    action = policy(observation)
    env.step(action)
env.close()
```

We further use the observation/action space objects from Gym, as well as the same seeding method and infrastructure (they were well done and very familiar to users).

Compliant environments wrap a general class (`AECEnv`). To allow for sufficient flexibility, environments only expose lower level attributes (dictionaries of values for all agents — `dones`, `infos`, `rewards`) and an observe method that takes an agent. These are then wrapped to provide the more general functions you see above by the base class, which allows for entirely new APIs to be efficiently added on top of PettingZoo environments should the need arise. We've done this ourselves with a secondary parallel POSG based API (that's very similar to RLlib's multi-agent API (Liang et al., 2018)) for a subset of the environments, due to special performance considerations.

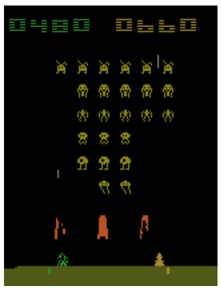
4 ENVIRONMENTS

Similar to Gym, we wanted to include popular and interesting environments within one package, in an easily usable format. Half of the environment classes we include (MPE, MAgent, and SISL), despite their popularity, have previously only existed as unmaintained "research grade" code, have not been available for installation via pip, have required large amounts of maintenance to run at all, and have required large amounts of debugging, code review, code cleanup and documentation to bring to a production-grade state. The Atari and Butterfly classes are new environments that we believe pose important and novel challenges to multi-agent reinforcement learning. Finally, we include the Classic class — classic board and card games popular within the RL literature.

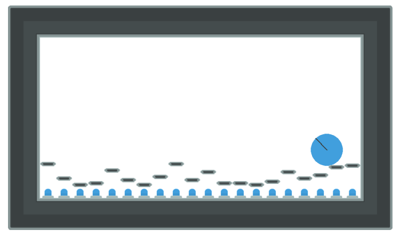
Atari

Atari games represent the single most popular and iconic class of benchmarks in reinforcement learning. Recently, a multi-agent fork of the Atari Learning Environment was created that allows programmatic control and reward collection of Atari's iconic multi-player games (Terry and Black, 2020). As in the single player Atari environments, the observation is the rendered frame of the game,

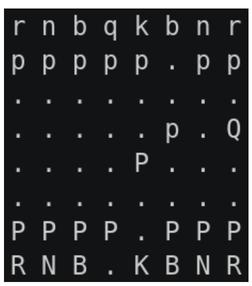
Figure 3: Example Environments From Each Class



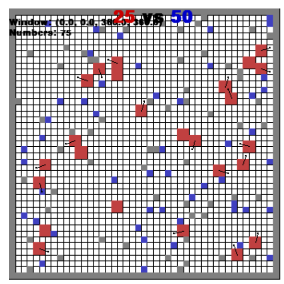
(a) Atari: Space Invaders



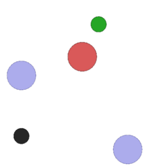
(b) Butterfly: Pistonball



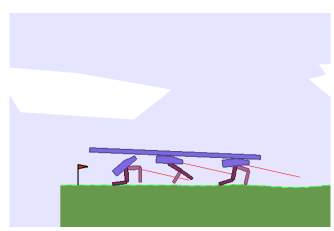
(c) Classic: Chess



(d) MAgent: Adversarial Pursuit



(e) MPE: Simple Adversary



(f) SISL: Multiwalker

which is shared between all agents, so there is no partial observability. Most of these games have competitive or mixed reward structures, making them suitable for general study of adversarial and mixed reinforcement learning. In particular, Terry and Black (2020) categorizes the games into 7 different types: 1v1 tournament games, mixed sum survival games (*Space Invaders*, shown in Figure 3a. is an example of this), competitive racing games, long term strategy games, 2v2 tournament games, a four-player free-for-all game and a cooperative game. For easy ROM installation, AutoROM, a separate PyPI package, can be used to easily install the needed Atari ROMs in an automated manner.

Butterfly

Of all the environments included, the majority of them are competitive. We wanted to supplement this with a set of interesting graphical cooperative environments. *Pistonball*, depicted in Figure 3b, where the pistons need to coordinate to move the ball to the left, while only being able to observe a local part of the screen, requires learning nontrivial emergent behavior and indirect communication to perform well. *Knights Archers Zombies* is a game in which players work together to defeat approaching zombies before they can reach the players. It is designed to be a fast paced graphically interesting combat game with partial observability and heterogeneous agents, where achieving good performance requires extraordinarily high levels of agent coordination. *Cooperative pong*, where two dissimilar paddles work together to keep the ball in play as long as possible, was intended to be a be very simple cooperative continuous control-type task, with heterogeneous agents. *Prison* was designed to be the simplest possible game in MARL, and to be used as a debugging tool. *Prospector* was included to intentionally be a very challenging game for conventional methods—it has two classes of agents, with different goals, action spaces, and observation spaces (something many current cooperative MARL algorithms struggle with), and has very sparse rewards (something all RL algorithms struggle with). It is intended to be an very difficult benchmark for MARL, in the same vein of Montezuma's Revenge.

Classic Classical board and card games have long been some of the most popular environments in reinforcement learning (Tesauro, 1995; Silver et al., 2016; Bard et al., 2019). We include all of the standard multiplayer games in RLCard (Zha et al., 2019): *Dou Dizhu*, *Gin Rummy*, *Leduc Hold'em*, *Limit Texas Hold'em*, *Mahjong*, *No-limit Texas Hold'em*, and *Uno*. We additionally include all AlphaZero games, using the same observation and action spaces—*Chess* and *Go*. We finally included *Backgammon*, *Connect Four*, *Checkers*, *Rock Paper Scissors*, *Rock Paper Scissors Lizard Spock*, and *Tic Tac Toe* to add a diverse set of simple, popular games to allow for more robust benchmarking of RL methods.

MAgent

The MAgent library, from Zheng et al. (2017) was introduced as a configurable and scalable environment that could support thousands of interactive agents. These environments have mostly been studied as a setting for emergent behavior (Pokle, 2018), heterogeneous agents (Subramanian et al., 2020), and efficient learning methods with many agents (Chen et al., 2019). We include a number of preset configurations, for example the *Adversarial Pursuit* environment shown in Figure 3d. We make a few changes to the preset configurations used in the original MAgent paper. The global "minimap" observations in the battle environment are turned off by default, requiring implicit communication between the agents for complex emergent behavior to occur. The rewards in *Gather* and *Tiger-Deer* are also slightly changed to prevent emergent behavior from being a direct result of the reward structure.

MPE

The Multi-Agent Particle Environments (MPE) were introduced as part of Mordatch and Abbeel (2017) and first released as part of Lowe et al. (2017). These are 9 communication oriented environments where particle agents can (sometimes) move, communicate, see each other, push each other around, and interact with fixed landmarks. Environments are cooperative, competitive, or require team play. They have been popular in research for general MARL methods Lowe et al. (2017), emergent communication (Mordatch and Abbeel, 2017), team play (Palmer, 2020), and much more. As part of their inclusion in PettingZoo, we converted the action spaces to a discrete space which is the Cartesian product of the movement and communication action possibilities. We also added comprehensive documentation, parameterized any local reward shaping (with the default setting being the same as in Lowe et al. (2017)), and made a single render window which captures all the activities of all agents (including communication), making it easier to visualize.

SISL

We finally included the three cooperative environments introduced in Gupta et al. (2017): *Pursuit*, *Waterworld*, and *Multiwalker*. *Pursuit* is a standard pursuit-evasion game Vidal et al. (2002) where pursuers and controlled in a randomly generated map. Pursuer agents are rewarded for capturing randomly generated evaders by surrounding them on all sides. *Waterworld* is a continuous control game where the pursuing agents cooperatively hunt down food targets while trying to avoid poison targets. *Multiwalker* (Figure 3f) is a more challenging continuous control task that is based on Gym's *BipedalWalker* environment. In *Multiwalker*, a package is placed on three independently controlled robot legs. Each robot is given a small positive reward for every unit of forward horizontal movement of the package, while they receive a large penalty for dropping the package.

5 DOCUMENTATION

Documentation is a fundamental part of a user-friendly software library. There's a tremendous amount of useful information about these environments, especially due to their diversity, so we sought to create as detailed documentation as possible, while designing it in a way to ensure it's still useful and approachable. PettingZoo includes comprehensive documentation for the API, the continuous integration tests, and each environment. A majority of popular libraries do not have extensive documentation. For example, OpenAI's popular Gym library only lists the observation space shape on each environment's documentation page. PettingZoo's documentation thoroughly explains each environment's observation and action spaces, and includes relevant information to help researchers. The goal is to allow people to compare environments easily, and for developers very rarely have to refer to source.

Our design for displaying so much information was inspired by Wikipedia's familiar and well-known layout. This is illustrated in Figure 4. All documentation is included in the supplemental materials to facilitate anonymous review.

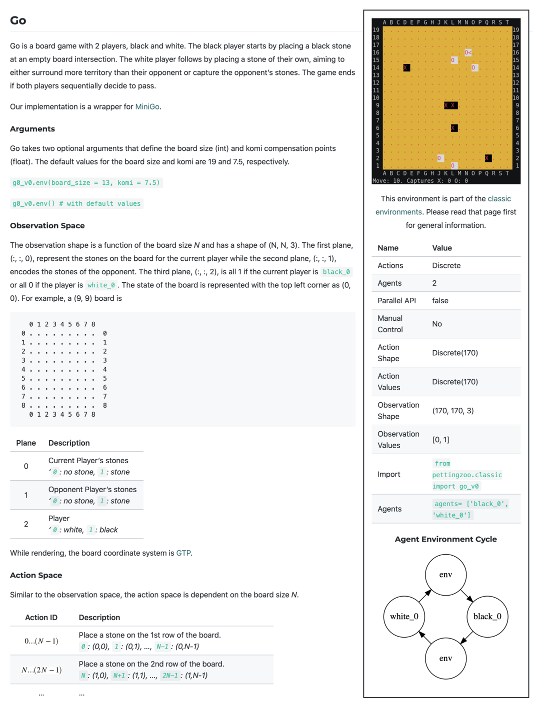
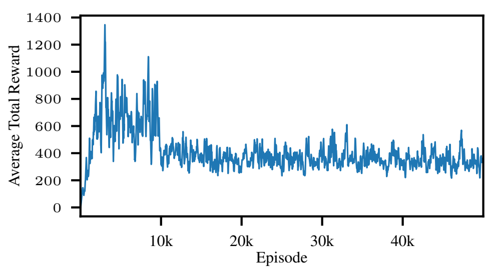


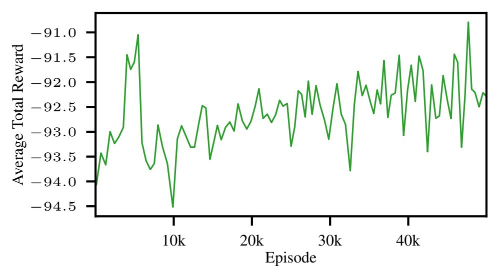
Figure 4: The beginning PettingZoo documentation for the Go environment, illustrating how we used the design metaphor of a Wikipedia page to include a large amount of detail in a manner that isn't overwhelming

6 BASELINES

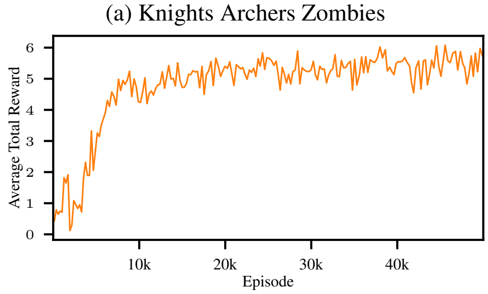
All environments implemented in PettingZoo include baselines to provide a general sense of the difficulty of the environment, and for something to initially compare against. We do this here for the Butterfly environments that this library introduces for the first time; similar baselines exist in the papers introducing all other environments. We used parameter sharing (Terry et al., 2020c; Gupta et al., 2017) with Ape-X DQN (Horgan et al., 2018), with RLLib (Liang et al., 2018). Our results are shown in Figure 5. Preprocessing and hyperparameter details are included in Appendix A. All preprocessing was done with the SuperSuit wrapper library (Terry et al., 2020a), which has recently added support for PettingZoo based multi-agent environments based. Code for the environments, training logs, and saved policies are available at `https://github.com/pettingzoopaper/pettingzoopaper`.



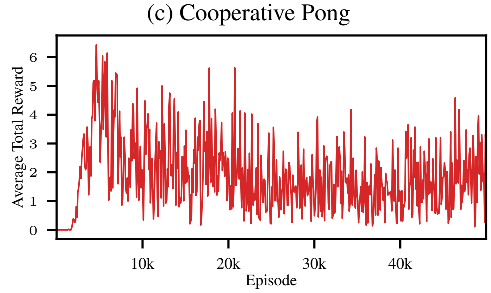
(a) Knights Archers Zombies



(c) Cooperative Pong



(b) Pistonball



(d) Prison

Figure 5: Total reward when learning on each Butterfly environment via parameter shared Ape-X DQN.

7 CONCLUSION

We introduced PettingZoo, a Python library of many diverse multi-agent reinforcement learning environments under one simple API, akin to a multi-agent version of OpenAI's Gym library.

Reinforcement learning systems have two aspects, the environment and the agent(s). Without a standardized environment base, research progresses by designing and building both the environment and the agent (as has been the case for MARL). The main contribution of PettingZoo is that it enables more research which focuses on the agent side of MARL by standardizing and democratizing the environment side, while at the same time incorporating many lessons learned from Gym. We hope that this allows for research in multi-agent reinforcement learning to accelerate and flourish.

We see three obvious directions for future work. The first is additions of more interesting environments under our API (possibly by the community, as has happened with Gym). Additionally, we envision a service to more easily allow different researchers' agents to play against each other in competitive games, leveraging the standardized API and environment set. Finally, we envision the development of procedurally generated multi-agent environments to test how well methods generalize, akin to the Gym procgen environments (Cobbe et al., 2019).

REFERENCES

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for AI research. *CoRR*, abs/1902.00506, 2019. URL http://arxiv.org/abs/1902.00506.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Y. Chen, M. Zhou, Ying Wen, Y. Yang, Y. Su, W. Zhang, Dell Zhang, J. Wang, and Han Liu. Factorized q-learning for large-scale multi-agent systems. In *DAI '19*, 2019.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Ben Goodrich. ale_python_interface. https://github.com/bbitmaster/ale_python_interface, 2015. GitHub repository.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018. URL http://arxiv.org/abs/1803.00933.

Łukasz Kidziński, Sharada P Mohanty, Carmichael Ong, Jennifer Hicks, Sean Francis, Sergey Levine, Marcel Salathé, and Scott Delp. Learning to run challenge: Synthesizing physiologically accurate motion using deep reinforcement learning. In Sergio Escalera and Markus Weimer, editors, *NIPS 2017 Competition Book*. Springer, Springer, 2018.

Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017. URL https://github.com/tensorforce/tensorforce.

Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. Emergent coordination through competition. *CoRR*, abs/1902.07151, 2019. URL http://arxiv.org/abs/1902.07151.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

G. Palmer. Independent learning approaches: Overcoming multi-agent learning pathologies in team-games. 2020.

Stefanie Anna Baby Ling Li Ashwini Pokle. Analysis of emergent behavior in multi agent environments using deep reinforcement learning. 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Sriram Ganapathi Subramanian, P. Poupart, Matthew E. Taylor, and N. Hegde. Multi type mean field reinforcement learning. In *AAMAS*, 2020.

Justin K Terry and Benjamin Black. Multiplayer support for the arcade learning environment. *arXiv preprint arXiv:2009.09341*, 2020.

Justin K Terry, Benjamin Black, and Ananth Hari. Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*, 2020a.

Justin K Terry, Nathaniel Grammel, Benjamin Black, Ananth Hari, Luis Santos, and Caroline Horsch. Agent environment cycle games. *arXiv preprint arXiv:2009.13051*, 2020b.

Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020c.

Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995. ISSN 0001-0782. doi: 10.1145/203330.203343. URL `https://doi.org/10.1145/203330.203343`.

Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE transactions on robotics and automation*, 18(5):662–669, 2002.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.

Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A toolkit for reinforcement learning in card games. *arXiv preprint arXiv:1910.04376*, 2019.

Lianmin Zheng, Jiacheng Yang, Han Cai, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *arXiv preprint arXiv:1712.00600*, 2017.

A Baseline Experiment Hyperparameters and Preprocessing

All of the environments were preprocessed in the following way: observations were resized to 84x84 images with linear interpolation, converted to grayscale, then normalized. This preprocessing was performed with SuperSuit (Terry et al., 2020a).

The graphically subtle environments (Knights Archers Zombies, Prospector and Cooperative Pong) had their observations processed with the RLlib default network: A convolutional layer with a 8x8 kernel, stride of 4, and 16 filters, followed by a convolutional layer with a 4x4 kernel, stride of 2, and 32 filters, followed by a convolutional layer with n 11x11 kernel, stride of 1, and 256 filters.

The graphically simple environments (Prison, Pistonball) were resized to 32x32 and flattened in addition to the above preprocessing. The observation was processed with a network with two hidden linear layers, 400 and 300 neurons wide, respectively.

RL method	Hyperparameter	Value
ApeX-DQN	`adam_epsilon`	0.00015
	`buffer_size`	400000
	`double_q`	`True`
	`dueling`	`True`
	`epsilon_timesteps`	200000
	`final_epsilon`	0.01
	`final_prioritized_replay_beta`	1.0
	`gamma`	0.99
	`learning_starts`	10000
	`lr`	0.0001
	`n_step`	3
	`num_atoms`	1
	`num_envs_per_worker`	4
	`num_gpus`	1
	`num_workers`	12
	`prioritized_replay`	`True`
	`prioritized_replay_alpha`	0.5
	`prioritized_replay_beta`	0.4
	`prioritized_replay_beta_annealing_timesteps`	2000000
	`rollout_fragment_length`	32
	`target_network_update_freq`	10000
	`timesteps_per_iteration`	15000
	`train_batch_size`	512

Table 1: Hyperparameters for ApeX DQN on each Butterfly environment.