Multi-agent Environment for OpenAI Lunar Lander

Siva Pabbineedi
Dept. of Computer Science
University of Colorado Boulder
Boulder, US
gapa2065@colorado.edu

Rishikesh Solapure
Dept. of Computer Science
University of Colorado Boulder
Boulder, US
rishikesh.solapure@colorado.edu

***Abstract*—There is a dearth of multi-agent environments in reinforcement learning (MARL) when compared to the plethora of options available for single-agent environments. PettingZoo library is currently at work to provide an intuitive API in MARL settings but it lacks some interesting environments that OpenAI Gym provides in single-agent settings. Our work creates a multi-agent lunar lander environment akin to the OpenAI lunar lander following the PettingZoo API. We have evaluated the environment we built successfully on a number of compliance tests that PettingZoo provides. We implemented agents using Deep Q-Networks and evaluated their performance in our environment.**

***Index Terms*—Reinforcement Learning, Multi-agent environment, Lunar Lander, OpenAI Gym.**

I. Introduction

Contrary to the already well-established OpenAI Gym for single agent reinforcement learning, standard APIs for multi-agent reinforcement learning (MARL) are only just beginning to appear. The development of single agent reinforcement learning research was greatly aided by the availability of common learning libraries like OpenAI Gym. PettingZoo appears to be one of the top libraries working to create multi-agent versions of Gym environments out of the small number of libraries working in multi-agent settings. However, because PettingZoo still lacks a variety of interesting environments from the Gym, it is only still a work in progress. Our work builds a multi-agent environment for the OpenAI Gym Lunar Lander. This would be a good addition to the PettingZoo library to help promote thorough comparisons in MARL research.

II. Related Work

The OpenAI Gym Brockman et al.(2020) provides a simple and widely used API that uses the partially observable Markov Decision Process (POMDP) conceptualization of reinforcement learning. But the Gym doesn't provide API or environments for multi-agent reinforcement learning problems.

Multi-agent API in RLlib Liang et al. (2018) uses the Partially Observable Stochastic Games (POSG) game model with dictionaries to hold actions, observations and rewards where agents act as keys. Everything in this model is simultaneous including the stepping of the agents and the assignment of rewards and observations. But this kind of API has trouble dealing with agent birth-death scenarios and games that are strictly turn-based like chess and poker.

OpenSpiel API Lanctot et al. (2019) uses the Extensive Form Games (EFG) game model which is more suited for turn-based games but their API is very complex unlike the Gym API. OpenSpiel's API also doesn't support continuous actions and only allows rewards at the end of the game which proves to be a big disadvantage for the learning process.

These two game models that are frequently used in MARL APIs – POSG and EFG may lead to unintentional bugs and are not strong choices for API creation as shown through case studies from Terry et al. (2021).So PettingZoo uses a game model called Agent Environment Cycle (AEC) for its API creation to achieve a superior conceptual fit to games simulated in code, unlike the previous APIs. Our work follows the PettingZoo API to implement the environment for OpenAI Lunar Lander to allow the agents to

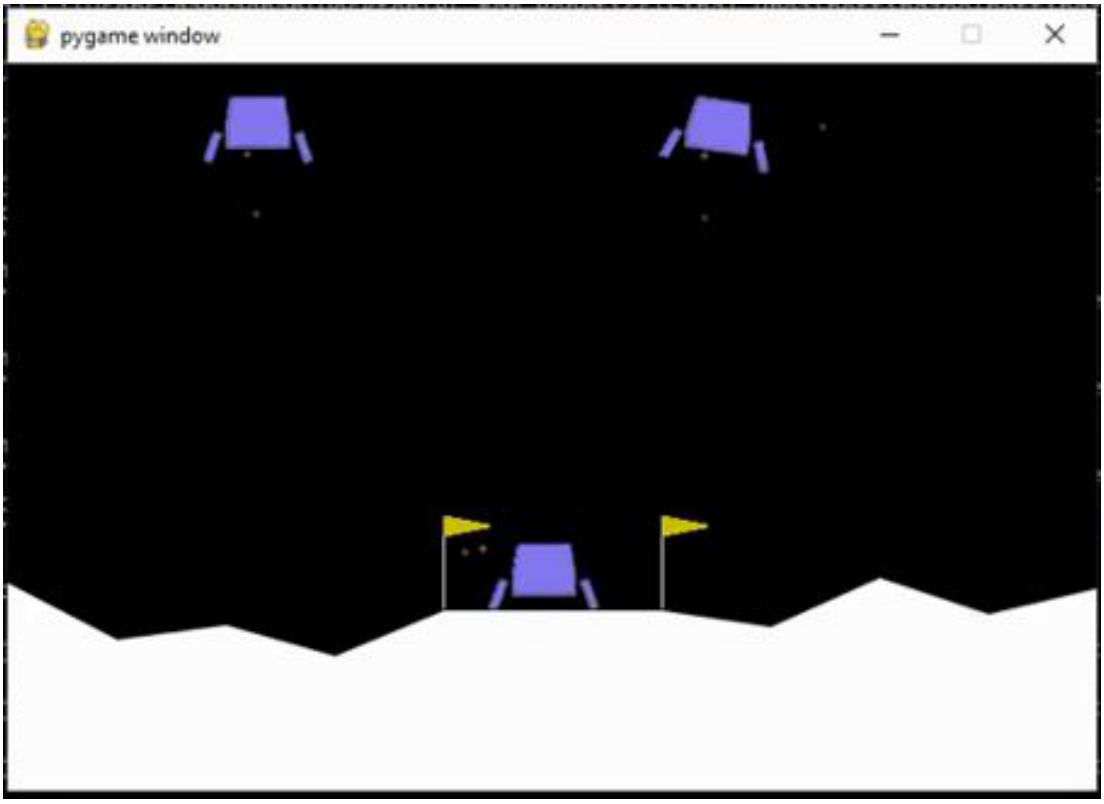

Fig. 1. Multi Lunar Lander when a agent is landing in a zone

sequentially get their observations and take actions. We also allow a version of the environment that follows the Parallel API from PettingZoo to accommodate simultaneous stepping like in POSG.

A multiple quadcopter OpenAI Gym-like environment based on a bullet physics engine was proposed by,Panerati, et al. (2021). A realistic collision and aerodynamics engine reinforce this multi-agent environment. However, in our method we don't have aerodynamics but we do consider a realistic collisions between multiple lunar landers using Box2D.

The naive DQN, which was put forth by Mnih et al. (2013), uses two fully connected layers and three convolutional layers to estimate Q values from images. DQN has been employed in a variety of applications, including portfolio management Gao et al., (2020), communications and network Luong et al., 2019, and autonomous control Quek et al. (2021). Due to simplicity and versatility, we used this algorithm to train our agents.

III. Methodology

A. Environment

In this section, we speak about how the environment is built. Most of our implementation draws from the OpenAI LunarLander and PettingZoo MultiWalker environments.

The agents are initially uniformly distributed at the top of the window with random linear and angular velocities which depend on wind power and turbulence respectively. The environment is made up of Box2D objects which are either static or dynamic. The moon is a static object with random terrain. Every lander is made up of a lander body, two legs and particles which are all dynamic bodies.

1) Observation Space: The observation space for any agent in this environment is a list of 12 unbounded values. They represent the horizontal and vertical lander positions, horizontal and vertical velocities, lander angle, angular velocity, leg ground contact values for both legs, and horizontal and vertical positions of the two nearest landers.

2) Action Space: The action space for any agent is the same as that of OpenAI Lunarlander. We support

both continuous and discrete action spaces. Discrete action space is made of four possible values for no action, fire left engine, fire main engine and fire right engine. The continuous space is a list of two values that represent the main and lateral engine fire values.

3) Collision Criteria: Collision criteria is the main thing that decides which objects collide with which other objects in the environment. OpenAI Lunar Lander has very a simple collision criteria where the lander collides only with the moon. This is usually implemented using category bits and mask bits. TABLE 1 indicates the collision criteria in the environment we made. TABLE 2 gives the values of the category and bit masks that achieve this collision criterion. Category bits in Box2d represent what category an object belongs to. The mask bits of an object is the bitwise OR value of all the category bit of the object that must collide with it. One caveat here is that we would not want a lander body to collide with its own legs. Box2D has a clever way to deal with this which is through GroupIndex. Assigning a unique negative group index to all the parts of a single lander would solve this problem. When we want two objects to collide, the bitwise AND of the category bits of object A and the mask bits of object B must not be equal to zero. The vice versa should not be zero as well. Getting the collision criteria right is probably the most important thing while building the interactions in an environment.

4) Reward Shaping: Reward shaping is done in an environment to make the learning process easier for the agents. We shape the rewards in a similar fashion to the OpenAI Lunar lander. We shape the reward to elicit a lesser distance to the landing zone, a slower horizontal and vertical velocity, a lesser horizontal tilt and to obtain ground contact on both legs. We also shape to get a higher reward when the agents maintain a safe distance from the other agents. This shaping acts like a gradient to take the landers to a desirable position. Just assigning a negative reward when agents collide, without proper shaping usually led to a high number of agent collisions.

5) Damage Criteria: The landers are considered damaged when the lander body gets in contact with any other object in the environment. The lander is also considered as damaged when any of its legs come in contact with any other object except the moon. The ground contact is made true as soon as any leg touches the moon surface and is made false if the leg bounces. There is a boolean value to control whether a lander should be removed when it is damaged.

A lander is considered as successfully landed when it lands on the zone or the moon and stays there without any movement. This is detected using the awake attribute of a Box2D object. As soon as a lander lands successfully, it is removed from the zone and its body disappears to make space for other landers to land. There are boolean values to control whether to terminate an episode when one of the agents gets damaged and whether to assign shared mean rewards to all the agents.

TABLE I
COLLISION CRITERIA

$Object_1$	$Object_2$	Collision
lander	lander	True
lander	leg	True
lander	particle	False
lander	Moon	True
leg	leg	True
leg	particle	False
leg	moon	True
particle	moon	True
particle	particle	False
moon	moon	True

TABLE II
CATEGORY AND MASK BIT VALUES

Object	Category Bits	Mask Bits
moon	0x0001	Lander|leg|particle|moon
lander	0x0002	lander|leg|particle
leg	0x0004	leg|lander|moon
particle	0x0008	moon

The "|" here respresents the Bitwise OR of the category bits of the object

B. Deep Q-Network

We used a DQN architecture with a multi-layer perceptron with two hidden layers of sizes 256 and 64. We used parameter sharing, so all our agents share the same model without appending the agent ID to the observation space.

IV. Experiments & Results

One can quickly track a model's development during training in supervised learning by comparing it to the training and validation sets. It can be challenging to accurately assess an agent's progress in reinforcement learning while it is being trained, but it is much more challenging to understand how the environment is performing and catching any bugs.

To evaluate an environment, the pettingZoo provides various environment complaince tests that allows one to check whether the environment is robust or not. We tested our environment on these compliance tests and it passed various tests that measure the parallel stepping capability and the deterministic behavior given a seed. We also executed the performance benchmark test and the save observation test whose results have to be manually examined. We have also evaluated the environment by training a DQN agent containing two layers by running it on 1,000 episodes.

TABLE III
Complaince Tests provided by PettingZoo

Test	Results
API Test	Passed
Parallel API Test	Passed
Seed Test	Passed
Max Cycles Test	Passed
Render Test	Passed
Performance Benchmark Test	Visually verified
Save Observation Test	Visually verified

We also have done some hyper-parameter tuning to maximize the reward that the agent is trying to get in the environment. These are the parameters that achieved the most promising result by far shown in TABLE IV

V. Conclusion

By successfully being able to land more than one agent in the landing zone and displaying its ability to cooperate with other agents to achieve the goal, we introduce a new multi-agent Lunar lander environment that functions well when trained on the DQN algorithm. Multi-agent Lunar Lander is functionally equivalent to other petting zoo environments and can be used in the same ways.

TABLE IV
Hyper-parameters of Best DQN Agent

Learning Rate	5e-4
No. of episodes	1000
Layer Size	256, 64
Max Cycle	2000
Average Reward	1,400
Buffer Size	1e5
Mini Batch Size	128

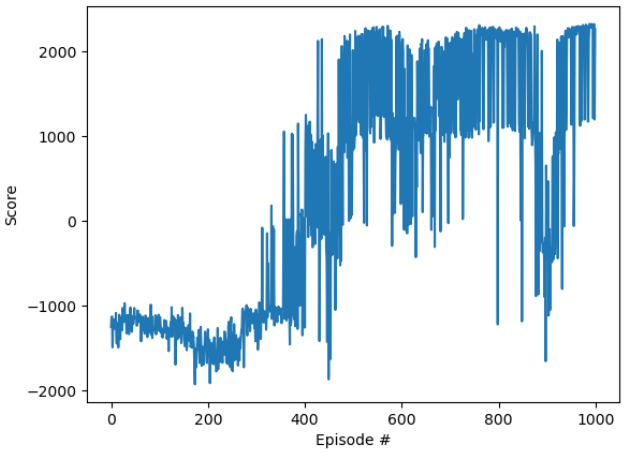

Fig. 2. Average Reward per Episode

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

Liang, Eric, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. "RLlib: Abstractions for distributed reinforcement learning." In International Conference on Machine Learning, pp. 3053-3062. PMLR, 2018.

Lanctot, Marc, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan et al. "OpenSpiel: A framework for reinforcement learning in games." arXiv preprint arXiv:1908.09453 (2019).

Terry, J., Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S. Santos et al. "Pettingzoo: Gym for multi-agent reinforcement learning." Advances in Neural Information Processing Systems 34 (2021): 15032-15043.

Panerati, J., Zheng, H., Zhou, S., Xu, J., Prorok, A., &

Schoellig, A. P. (2021, March). Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 7512-7519). IEEE

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

Gao, Z., Gao, Y., Hu, Y., Jiang, Z., Su, J. (2020, May). Application of deep q-network in portfolio management. In 2020 5th IEEE International Conference on Big Data Analytics (ICBDA) (pp. 268-275). IEEE.

Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y. C., Kim, D. I. (2019). Applications of deep reinforcement learning in communications and networking: A survey. IEEE Communications Surveys Tutorials, 21(4), 3133-3174

Quek, Y. T., Koh, L. L., Koh, N. T., Tso, W. A., Woo, W. L. (2021). Deep Q-network implementation for simulated autonomous vehicle control. IET Intelligent Transport Systems, 15(7), 875-885.