Team Slime Volleyball
Project Report

Oleksandr Stopchak
90627
oleksandr.stopchak@tecnico.ulisboa.pt

Manuel Goulão
91049
manuel.silva.goulao@tecnico.ulisboa.pt

Mariana Ferreira
86478
mariana.bento.ferreira@tecnico.ulisboa.pt

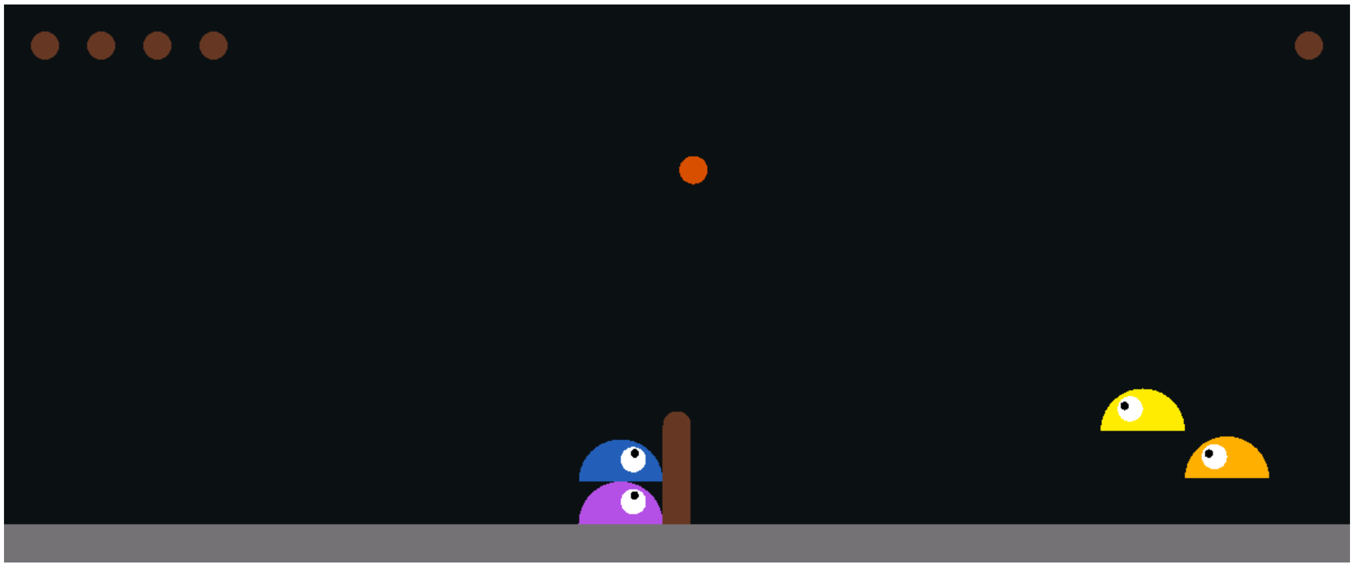

Figure 1: Slime Volleyball game with teams of 2 players

ABSTRACT

The aim of this project was to implement our own version of the game Slime Volleyball, where we have teams of two players. We test four distinct versions of the PPO algorithm, where the agents have different rewards for performing specific roles. Our goals were to assess how the different models performed in a multi-agent system, to analyze the agents' behaviors and patterns that arised during training and, lastly, to compare how the different models performed against one another. The video deliverable requested can be found here[1].

KEYWORDS

reinforcement learning, multi-agent, slime volleyball, adversarial games

1 INTRODUCTION

For the curricular unit of Autonomous Agents and Multi-agent Systems, we were asked to choose a project that addressed a real-world problem using intelligent systems. As such, our project focused on a multi-agent environment where coordination between teammates was the key to reach a common goal, which, in this case, consisted of scoring points. We chose an already implemented environment called *Slime Volleyball* [5] and modified it to fit our own goals, creating enough complexion for it to become a coordination problem. In of itself, this was not a difficult task, since all it required was adding two more agents to the playing field, one on each team. Yet, this apparently simple task added an exponential amount of detail to the environment, also making the process of training the agents exponentially longer and more difficult. Given our limited time as students to focus on one project at a time, this was an interesting yet ambitious project, but nevertheless, we did it anyway.

Although it was advised to send all our files (including the videos) for this project in a zip through Fenix, we decided that it was best to upload them to a cloud drive and link the videos instead. This way there is no quality loss on our side, and the uploaded file is much lighter for the evaluating professor.

In the Section 3, we describe the environment we used, especially the changes made to the original Slime Volleyball game. In section 4 we explain the method that we used to train the agents that we test in the evaluation, self-play. In section 7 we show and specify all the agents we trained and their characteristics. Finally, in section 8 we will present the results of the evaluations we made with the agents.

2 RELATED WORK

The majority of work published in recent years about reinforcement learning has been about single-agent learning, while multi-agent environments are less commonly investigated. Some recent

[1]https://drive.google.com/file/d/1-oFUVFExdCrVahZ$_Y1XVJuYaUADwC06c/view?usp=$ *sharing*

work in multi-agent reinforcement learning approaches the problem through Inverse Reinforcement Learning [7][10]. Another recent paper proposes a Soft Team Actor-Critic to solve the problem of coordination without any prior knowledge about the domain [3]. Also, a release of the ML-Agents Toolkit for Unity included a feature that added the capability to train agents in adversarial environment, just using self-play and single-agent Reinforcement Learning algorithms with no role assignment [4].

3 ENVIRONMENT

Adding additional agents to our environment required some adjustments, namely changing the observation space and implementing collisions between elements of the same team.

Initially, when there was only one slime on each side of the pitch, our observations consisted of the positions and velocity of the agent, the ball and the opposing slime. When we added the second element to each team, the size of the observation space almost duplicated, since we were including information regarding the agent's teammate and both opposing agents as well. This not only added complexity to our already intricate environment, but also made it almost impossible to train the agents, due to having an *enormous* amount of possible states. To solve this issue we decided to limit the observation space by excluding all information regarding the enemy team, keeping only the agent's info, plus the ball and the teammate's info. These changes allowed the agents to train ever so more quickly. This will be better explained in Section **Simplified Environment**.

Regarding the collisions, we had to ensure that agents didn't overlap and we decided that, when an agent ran into its teammate, the latter would not be affected by the impact and would remain in the same place. We also had to take into consideration that the agents can stay on top of one another, which happens when they share the same x coordinates. In this particular case, the agent on top becomes able to jump higher, so we slightly raised the height of the net in the middle of the field.

4 SELF-PLAY

In Reinforcement Learning, agents try to solve a task by learning behaviours that maximize their reward. In the Slime Volleyball environment, which is an adversarial game, the agents should learn to play against the other team in order to maximize their reward, but we also want them to play the game well, so that they are able to win against any team they may face. A training technique used in these types of adversarial games is called self-play.

When training using self-play, the agents of the team being trained start playing against a random policy. After X timesteps, we evaluate the agent performance and, if the average reward in those X timesteps is bigger than some threshold Y, then a copy of the agent is created and this copy is then used to replace the previous adversary.

Self-play allows the agents to always play against a team with the same level of quality, which forces them to keep improving, since getting rewards will become slightly more difficult, timestep after timestep.

Another option we had considered was to start playing against the baseline model and only after the agent started beating it would it start playing against its own best version, but we decided to start with the random policy.

Table 1: PPO parameters.

Parameter	Value
Timesteps to update (Horizon)	4096
Epochs	10
Discount (gamma)	0.99
Clipping parameter	0.2
Actor learning rate	0.0003
Critic learning rate	0.001

5 PPO

The PPO implementation was obtained from the public repository *PPO-PyTorch* [1]. Most of the baseline libraries publicly available are very inflexible and adapting them to train more than one agent would require us to change a lot of code. For this reason, we decided to go with a very simple implementation that would give us the flexibility we needed, like, for example, adding roles. The parameters used are presented in Table 1. In order to assess if the implementation we used was working as expected and understand how the changes we had made to the environment affected the number of timesteps that the PPO took to converge, we made a few initial tests, as described in the next two subsections.

5.1 PPO train vs. Baseline 1v1

A single agent using a PPO policy trains against the Baseline policy. This allowed us to understand if the PPO implementation we used had similar results to the one the Slime Volleyball repository uses. In Figure 2, we observe that, after a few million timesteps, the agent starts closing the gap to the Baseline and, in the end, it converges to an average score of -1.0. This is a satisfactory result, since it converges to an average reward close to zero and in the same interval of timesteps of the repository's PPO. On top of this, reproducibility in Reinforcement Learning is know to be difficult since even small changes to parameters or even seeds can create different final results [6] .

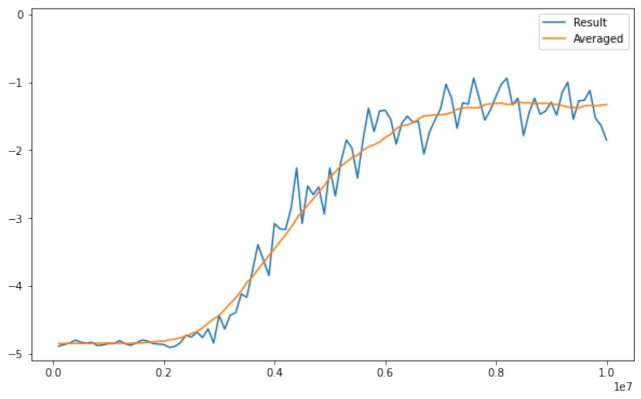

Figure 2: Training against Baseline in a 1v1

5.2 PPO train vs. Baseline 2v2

We now knew that we had a working implementation of the PPO. But the new 2v2 environment that we created could be more difficult for the agents to learn. With this test, we were able to assess if our environment was in fact more difficult to learn in. The results, in Figure 3, clearly show that our agents are not learning. The two main hypotheses to explain this were that the environment might be too difficult to learn in and that the collisions between teammates might be creating some stochasticity.

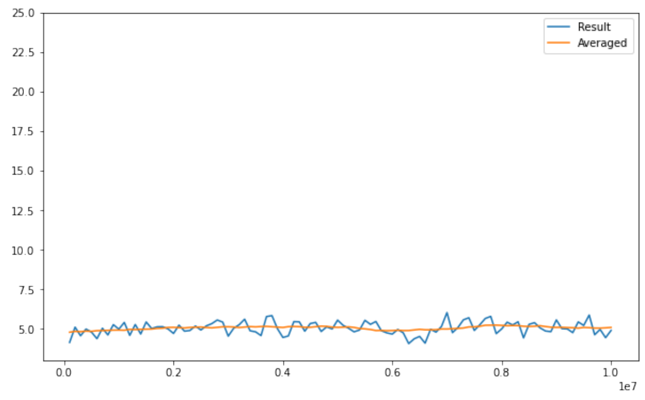

Figure 3: Training against Baseline in a 2v2 with survival reward

6 SIMPLIFIED ENVIRONMENT

One of the hypotheses that could explain the bad performance of the agents after the training phase was the growth of the observation space. Thus, we tested removing less important parts of the observation space, namely the positions and velocity vectors related to the opposite team. After simplified, the observation vector became $< x, y, vx, vy, team_x, team_y, team_vx, team_vy, ball_x, ball_y, ball_vx, ball_vy >$ Another modification made to the environment was changing the reward function, as the rewards in the original environment were very sparse, which made the learning more difficult. To improve this, we changed the reward function to reward the agents not only for scoring a point, but also for playing each point for a longer time, which creates a nice heuristic in order for the agents to learn to keep the ball in the air. Running the previous test (now using the simplified observation space and the new reward function) shows much better results, as emphasized by Figure 4. The learning is not as fast as in the 1v1 test, but considering that we now have 2 players on each team we believe the results are good enough for us to carry on.

7 AGENTS

In total, we have 4 different teams of agents for this project: *PPO*, *PPO-AD* (Attacker/Defender), *PPO-FloorIsLava* and *PPO-Leader*. As their names emphasize, all these agents use the PPO algorithm. Each team is composed of two agents independent of each other and from this point onwards we will be using the simplified environment described previously.

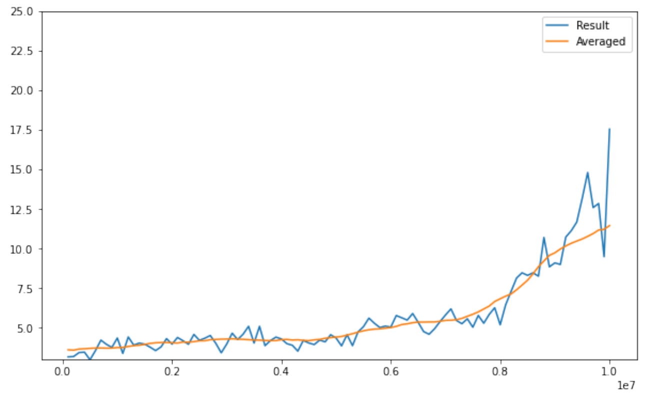

Figure 4: Training against Baseline in a 2v2 with survival reward and observation space simplification

In this section, we will explain how we implemented all these teams and how we have trained them using self-play.

7.0.1 PPO. The PPO team is very straightforward: each agent uses the PPO algorithm, having access only to their own observation and not knowing what the teammate is doing or wants to do. In Figure 5 we can recognize that the agent is not learning very well, since the reward after each game remains the same through the training and assuming that better agents play longer points. Remember that this reward is the survival reward, meaning that the longer the points the bigger the reward.

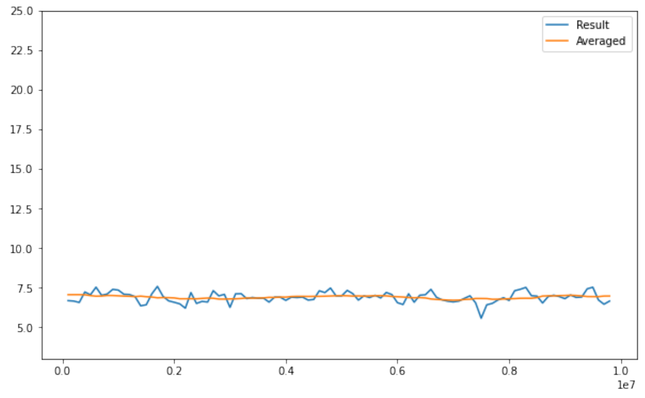

Figure 5: Training *PPO* with self-play

7.1 PPO with Roles

One of the problems of using PPO in a multi-agent environment where the agents can interact with each other is the lack of capacity of the agents to communicate and/or coordinate. To solve this issue we developed several strategies.

Before we dive into the different strategies, we first need to explain the concept used, which, in this case, was that of roles. Here, any given agent within a team must have a specific goal in mind at every timestep. And, since having each agent blindly committed to a goal is a bad idea in such a dynamic environment, agents from the same team communicate with each other frequently - always trying to figure out the best distribution of roles possible, given the state that they're in. To do this, each pair of roles shares a potential function, which helps the former decide their next goal. To train the agents, an intrinsic reward was used, i.e. every reward is modified according to how 'enjoyable' it is for the specific role [9]. For each of the following pairs of roles, this was done for approximately 10M iterations, in the hopes of getting the agents to always choose the best action according to their intentions.

7.1.1 Attacker/Defender. For our first strategy, we used the classic attacker/defender combo. The gist of these roles is to have one agent trying to score a point against the other team, while the other one gets into a defensive position, so they occupy as much of their playing field as possible for when the ball gets back to it.

Under these circumstances, we had to pay attention to two specific things when training the agents:

- The distance to the ball;
- The distance between teammates.

For the former point, we reward the attacker for being closer to the ball, and the defender for being further away, with some exceptions, to make this more interesting. First, for simplicity, let's define the attacker as A and the defender as D. If the agent is an A and the ball gets close to D within a certain threshold, then A is rewarded for staying still rather than keep getting closer to the ball, until the roles get switched, and vice-versa.

For the latter, we took inspiration in the current pandemic and rewarded the agents for taking social distancing seriously.

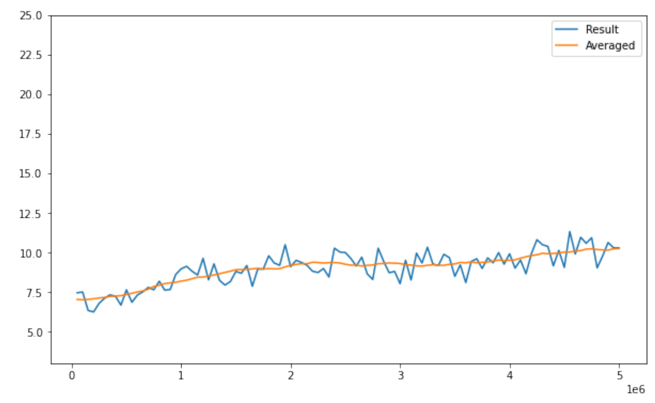

Figure 6: Training *PPO-Attacker/Defender* with self-play

Behavior: For a quick overview of the behaviour that we will be describing next, please watch this video[2]. As you can see, the agents are behaving somewhat as described before: once the ball is on their playing field, one of the agents tries to follow it with the intent of sending it to the other side, while the other one maintains a healthy distance in case the ball doesn't go immediately to the other side and instead moves towards him. This last detail was not exactly what we had hoped for in the beginning, but it actually turned out way better than our initial plan, since this environment is stochastic and most of the time the ball doesn't move necessarily to where the agents get the best reward (score a point).

As a side note, one other curious aspect about the final result is that the agents learned some interesting tactics, namely staying on top of each other near the net, trying to block the ball as soon as they can.

7.1.2 FloorIsLava. Another scenario we tried was "the floor is lava", or top/bottom, where one of the agents (top, or T) tries to stay, as much as possible, on top of his teammate, being encouraged to only touch the ball when this happens. In turn, the teammate (bottom, or B) tries to position itself close to the ball.

The main concerns while training were:

- The distance of B to the ball
- Ensuring T only touched the ball when placed on top of B
- The distance between T and B

The first concern is addressed by rewarding B for reducing its distance to the ball and punishing it when this distance increases. To ensure T places itself on top of B, we checked their x coordinates, giving T a significant reward whenever these matched. The distance between T and B is also important, since, if they are too far apart, it becomes impossible to reach the desired top position in the next timestep. Thus, we wanted to encourage T to move closer to B (and, consequently, also closer to the ball).

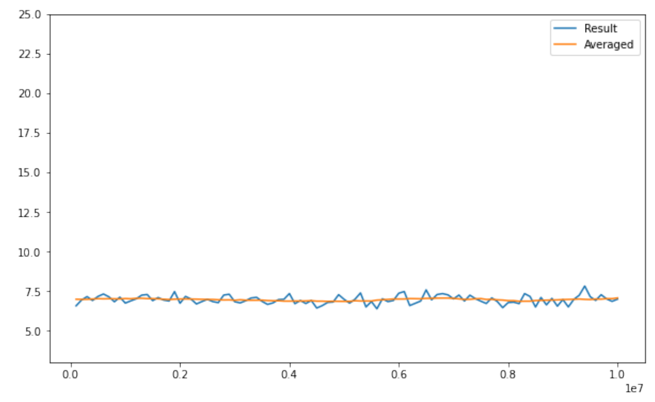

Figure 7: Training *PPO-FloorIsLava* with self-play

Behavior: To achieve the desired results, we had to test different reward values. In a preliminary testing phase, although the agents were paying little attention to the ball (thus failing the primary goal of the game), we were able to notice an interesting behaviour pattern: the agents would move to the edges of the pitch and try to remain there. The reasoning behind this, we believe, is that, since the agents can't move past the game limits, it's easier for them to become aligned on top of each other. A quick overview can be seen in this video[3].

Since it was still necessary to improve the way the agents dealt with the ball, we had to make some modifications and this pattern

[2]https://drive.google.com/file/d/1rGteYkO6vBvoDl0WuCV0XkGfoJGh0TiB/view?usp=sharing

[3]https://drive.google.com/file/d/1QcF4r9TjUb4Mx7DbMFLM8OUpU-BZUieW/view?usp=sharing

was lost. However, a similar one emerged, but this time next to the net. We have also noticed that the agents seemed to favor being next to the teammate more than being on top of it, even though the reward for being close was smaller than that for being above. These differences can be seen in this video[4].

7.2 Communication

7.2.1 Leader. Until this point, we had tried different roles to achieve coordination, but we had not yet explored communication between the teammates. Our solution to achieve uni-directional communication between the team's agents is called *PPO-Leader*, which was somewhat inspired by *Feudal Q Learning*. In this team, *agent 1* is considered the leader and has two functions to decide his own actions and give orders to agent two. These orders are simply the action that the leader wants *agent 2* to make. During the training phase, agent 2 will try to learn how to satisfy the leader's orders, while the leader will try to maximize the environment's rewards by taking good actions and giving good orders.

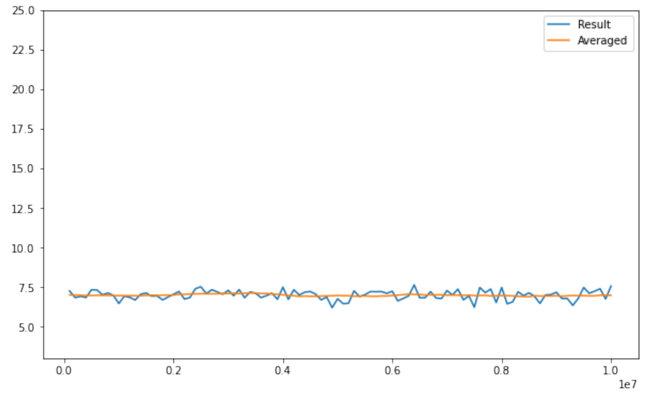

Figure 8: Training *PPO-Leader* with self-play

Behavior: This team did not have any particular behaviour like the previous ones, although it is possible that in a different environment (for instance, one with obstacles) this team would require less timesteps to adapt to it. This is due to the fact that agent 2 is just following orders and, for that reason, only the leader would need to partially relearn the environment.

8 EVALUATION

After training all the agents for 10 million timesteps, we put each of the teams playing against the Baseline. In this section we will present the results of this evaluation.

As we can observe in Table 2 all teams performed very poorly. The *PPO-Attacker/Defender* was the only team able to win games and was also the best team in terms of mean score. This can also be seen in Figure 6, where, although the reward gained does not rise as sharply as in Figure 4, it still rises, which means we can expect better performance over time. This result follows what we were expecting after observing the agents play during and after the learning phase.

[4]https://drive.google.com/file/d/1flwOwyJtscsVP7fwfJdzGB3ZPzu1Znv1/view?usp=sharing

Table 2: Teams evaluation against Baseline

Team	Mean	Standard Deviation
PPO	-4.372	0.895
PPO Attacker/Defender	-4.160	1.056
PPO Floor is Lava	-4.306	0.901
PPO Leader	-4.32	0.842

9 DISCUSSION

The roles required us to do a lot of reward engineering in order to make the agents behave the way we wanted. Also, the slow training/tweak loop did not allow us to fully explore more values for the intrinsic rewards.

The way we made the agents learn with self-play might be problematic, considering that the agents spend many timesteps playing against very bad policies. An alternative, as we have already mentioned, would be to start training against the Baseline provided with the original Slime Volleyball repository, but that would defeat the idea of starting from scratch. Moreover, the Baseline was trained on a 1v1 environment, so it doesn't take advantage of the strategies available in a 2v2 environment.

During the project, we used PPO as the basis for all our experiments and teams. Current state-of-the-art model-free algorithms like PPO are very sample inefficient, as they take several million timesteps to learn tasks [8][2]. In our case, this meant spending several hours to train each version, meaning that tweaking the agents could take a lot of time. Even in the 1v1 experiments, the PPO agent took several million timesteps to start increasing the reward.

One common behaviour we observed in the agents was their lack of capacity to generalize well. This was obvious in situations where an agent in a similar position relative to the ball but in different coordinates in the environment would behave very differently.

10 FUTURE WORK

After finishing all of our implementations and evaluations, we were not satisfied with some of the results. Most of these could be improved by having more time to train and tweaking agents, but since each training session took multiple hours to have any viable results, and, sadly, this project having a due date, it just wasn't feasible or practical exploring all these new ideas and twists that we had in mind. That being said, some of the most notable ideas that we planned on adding but were not implemented are:

- Adding static/movable obstacles to the playing field;
- Adding another communication game mode, where the agents had to touch the ball at least once before trying to score a point;
- Adding power ups that could be obtained if the ball touched them.

11 CONCLUSION

To summarize, we would like to mention that, although not all implementations were perfect and had expected results, there are still lots of conclusions that can be taken to improve our work, such as, what is accomplishable in a similar environment to ours, using PPO or any other comparable algorithm. We also showed that reinforcement learning is still very slow and computational intensive, which makes research about the topic difficult for groups with less computational resources, as can be seen by the amount of time that our agents took to train. Within our implementation, we found out that the role Attacker/Defender had really promising results and should be looked into a bit further. And finally, we would like to think that we made some contributions that allow us to better understand how reinforcement learning methods perform in a multi-agent context.

REFERENCES

[1] Nikhil Barhate. 2021. Minimal PyTorch Implementation of Proximal Policy Optimization. https://github.com/nikhilbarhate99/PPO-PyTorch.

[2] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617* (2018).

[3] Andrea Celli, Marco Ciccone, Raffaele Bongo, and Nicola Gatti. 2019. Coordination in Adversarial Sequential Team Games via Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:1912.07712* (2019).

[4] Andrew Cohen. 2020. Training intelligent adversaries using self-play with ML-Agents. https://blog.unity.com/technology/training-intelligent-adversaries-using-self-play-with-ml-agents

[5] David Ha. 2020. Slime Volleyball Gym Environment. https://github.com/hardmaru/slimevolleygym.

[6] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning that Matters. *CoRR* abs/1709.06560 (2017). arXiv:1709.06560 http://arxiv.org/abs/1709.06560

[7] Xiaomin Lin, Peter A Beling, and Randy Cogill. 2014. Multi-agent inverse reinforcement learning for zero-sum games. *arXiv preprint arXiv:1403.6508* (2014).

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

[9] Satinder Singh, Andrew G Barto, and Nuttapong Chentanez. 2005. *Intrinsically motivated reinforcement learning.* Technical Report. MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE.

[10] Lantao Yu, Jiaming Song, and Stefano Ermon. 2019. Multi-agent adversarial inverse reinforcement learning. In *International Conference on Machine Learning*. PMLR, 7194–7201.