RallySim : Simulated Environment with Continuous Control for Turn-based Multi-agent Reinforcement Learning

Shun Yoshikawa[1], Yusuke Mukuta[1,2], Takayuki Osa[1,2], and Tatsuya Harada[1,2]

[1]The University of Tokyo
[2]RIKEN Center for AIP
{yoshikawa, mukuta, osa, harada}@mi.t.u-tokyo.ac.jp

***Abstract*—Multi-agent reinforcement learning has been considered a promising approach to train agents for tasks that involve cooperative and competitive interactions between players. Several games have already been used as multi-agent problems, and studies have proven that existing reinforcement learning methods along with the multi-agent architecture can get an agent with satisfied behavior. However, the difficulty of the existing multi-agent tasks is modest in that they usually have as small and discrete action spaces as game controller inputs. There need to be more multi-agent tasks with complicated continuous control. In this paper, we propose a customized multi-agent environment, RallySim, where agents play a task that is inspired by rally sports. In this task, two player robots placed on a court are supposed to hit a ball with their end effectors to perform multiple exchanges of the ball between them. It entails both the learning of motor-skills and strategies, a factor that the existing tasks do not contain. We use hierarchical reinforcement learning for training an agent for the RallySim task, and the evaluation results show that outperforms agents trained with ordinary architecture.**

***Index Terms*—Multi-Agent Systems, Deep Reinforcement Learning**

I. Introduction

Reinforcement learning (RL) has demonstrated remarkable performance in diverse fields, including acquisition of strategy [1], [2], robot manipulation [3], [4], and motion generation [5], [6]. Among the countless problems of RL, training an agent for situations where multiple players interact with each other is one of the promising fields. If agents can be trained in such multi-player settings, it widens the range of problems RL can solve up to more complicated, realistic kinds in which there are several humans, and interaction among them makes the situation often unpredictable. Multi-agent reinforcement learning (MARL) is one of the approaches to tackle these problems, and many studies have been conducted to solve multi-agents tasks efficiently [7], [8]. Along with exploring various kinds of MARL, new tasks with multi-agent setting have appeared and been utilized as benchmark tasks of MARL.

Although many of the existing benchmark tasks are good in quality and played an imperative role in developing the field of MARL, we believe that their difficulty leaves some room to be desired, and there can be more tasks with more difficult settings. One thing the existing tasks have in common is the small size of their action space. Many benchmark tasks for MARL today are inspired by video games such as StarCraft, and DOTA2, to name a few. When an agent infers its action in these tasks, it just chooses a combination of input buttons, which can be represented only by a small group of discrete numbers. If the action space of a task were not described by controller inputs, it would still be mostly discrete and would not give enough complication.

In this paper, we choose a rally of sports as a new MARL task. We suppose that a rally involves two players on a designated playing field with a target object that the players aim to hit with their racket. The novelty of this task lies in that being able to play a rally entails two types of learning. One is to device where to hit the ball either to maintain the rally for as long as possible or to make shots that give an advantage to the player during a game. The other is to optimize its continuous movement to properly hit the ball in accordance with its intention. Having this two-stage process makes the rally problem distinct from any other MARL task that already exists.

We develop a RallySim, a simulated environment where robot players move with continuous action space and hit a ball to realize a rally. We conduct training in the simulation to find a way to get a model that displays a desired performance in the rally task. Through the experiments, we found that goal conditioned method with a hierarchical reinforcement learning (HRL) architecture works well in acquiring both the ability for decision-making and optimized motor skills [9], [10], [11].

II. Related Work

In this section, we introduce some of the existing MARL tasks and compare them with RallySim. We compared RallySim with five tasks: MAgent2 [12], SMAC (StartCraft Multi-Agent Challenges) [13], ML-Agent Soccer [14], DOTA2 [15] , and the soccer task from DeepMind [16]. The action spaces of the first three tasks are all discrete with modest size (ranging from several to dozens). Each action in these spaces directly represents certain commands, so in these tasks, agents only need to focus on optimizing their strategy of which command to choose during each situation. Our RallySim distinguishes itself from tasks with discrete action spaces in that agents need to learn how to move the body joints of the robots in addition to the strategic factor of the task, which is to choose where they should return a ball. DOTA2 also has a discrete action space, but its size grows up to 8000 to 80000 actions, making the task relatively complicated to learn. RallySim can still considered a task that is distinct from any tasks with discrete action spaces, including DOTA2, since its action space is continuous and, therefore, provides new factors of complication during training.

Among the five tasks shown in the table, the soccer task recently released from DeepMind has the same level of difficulty as RallySim. Players deployed inside the simulation are physically based human models that learn to perform basic skills in soccer such as dribbling, following targets, and kicking a ball to a goal.

We contend one major difference between the soccer task and our RallySim is that the decision-making process in RallySim is rather turn-based, while that in the soccer task seems to be ceaseless and dynamic. When a rally is going, a player makes a decision only at the moment its opponent makes a shot and the ball starts to fly toward the player's side. The player then moves based on the decision it made, which is fixed until it hits the ball. Agents sample less experience through the training than they do in such tasks that the decision is sampled at each step because the frequency at which a player's decision is sampled is low. This turn-based feature of the rally task adds another layer to its difficulty as the training efficiency deteriorates.

III. Creation of RallySim

RallySim is created using Unity with version 2022.3.7.f1 [17]. Unity ML-Agents are utilized for building RL modules in the simulation architecture. We provide two types of simulations, each of which is designed in 2D and 3D, respectively. Most of the results shown in the following sections are from the 2D environment. We made and executed training in the 3D environment to suggest the outlook of this research.

A. Overview of Simulations

The customized simulations model a game of badminton, a kind of sport that involves rallies. They consist of three main components: a court, players, and a ball. Players are robots with multiple joints, each of which is placed on either of the two sides of the court. When a ball approaches one of the players, it tries to move its body joints to realize the desired movements for hitting the ball with its end effector or a racket. When the racket hits the ball, proper force is applied to it based on the collision, shooting it over to the opposite side. Then, the other player is expected to return the ball. Repeating this process allows the players to maintain a rally, an exchange of the ball between them. During the simulation, players are supposed to have as long rallies as possible; and this is the objective of a learning agent.

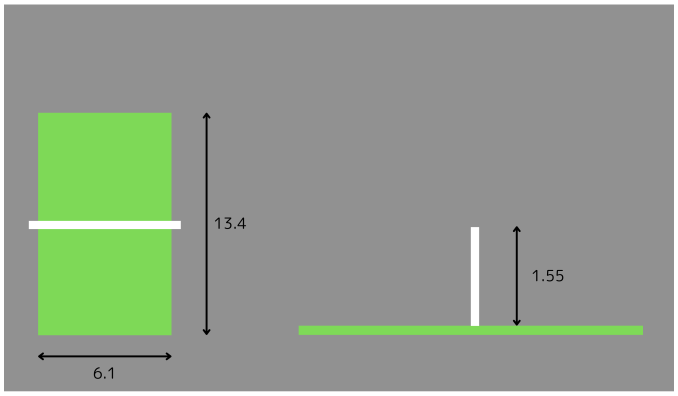

Fig. 1. The concept drawing showing the main components of the court used in the simulation environments (universal in all types of simulated environments).

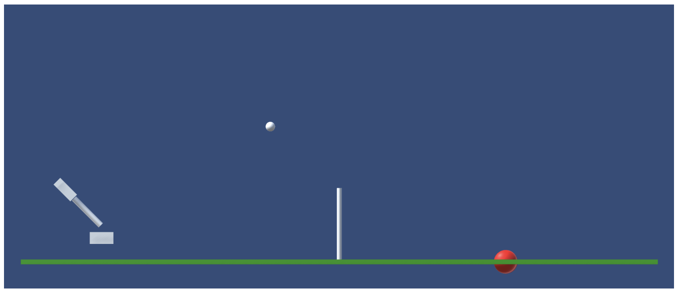

Fig. 2. A snapshot of the simulation environment with a robot with a low degree of freedom (SIMPLE).

B. Simulation Landscape

In both types of simulations, a rectangular plane resembles a badminton court. A white, wall-like object stretches across the plane to divide it into two sides with equal areas. This object models a net of a badminton court. A collider is attached to the net, which means when a ball flies in a trajectory that penetrates the net, it will be blocked by the net and cannot go through it. Fig. 1 shows the scale of the court in Unity units. 1 Unity meter corresponds to 1 meter in the real world.

C. Player in 2D Environment

Fig. 2 is a snapshot of the 2D environment, which we call SIMPLE. Inside the simulation, an arm-shaped robot exists and is supposed to learn to hit a white ball with its end effector. The robot only moves in the horizontal direction and rotates with one axis that is perpendicular to the simulation plane. With degree 0 being the robot standing straight, it can rotate from negative 85 degrees to positive 85 degrees. At each simulation step, the robot moves either forward or backward to catch up with a coming ball, the white sphere shown in Fig. 2, and performs a one-way swing by rotating with the axis.

Tables I and II show the state space and the action space of SIMPLE, respectively. The value of each element in a vector ranges from -1 to 1, which is scaled appropriately when delivered to the environment simulator as input.

TABLE I
STATE SPACE IN THE SIMPLE

Property Name	Vector Size
Robot Position	3
Arm Orientation	4
Ball Position	3
Ball Velocity	3
Target Position	3
Total	16

TABLE II
ACTION SPACE OF THE SIMPLE

Property Name	Vector Size
Target Orientation	1
Force	1
Value of Movement	1
Total	3

D. Player in 3D Environment

Fig. 3 depicts the 3D simulation, HUMAN-LIKE. The difference between SIMPLE and HUMAN-LIKE is whether the ball flies and the robot moves in a 3D space.

The body of a player is derived from Unity's machine learning toolkit, ML-Agents [14]. ML-Agents provide different kinds of sample environments that are useful when users try to incorporate RL in their development projects, one of which is Walker, a task in which a model resembling a human body learns to walk with its legs (snapshot shown in Fig. 4).

We use the human body model used in the ML-Agents Walker task and adapt it to our customized task environment. In HUMAN-LIKE, the player's lower body is detached, so we are not concerned about training the agent to walk, which is a difficult problem and not related to the main part of our research. Table III lists the joints of the robot with constraints on the directions they rotate. For each joint, 14 elements are added to the state space of the environment, as shown in Table IV. The whole state space and the action space are shown in Tables V and VI.

E. Behavior of a Ball

As the word already appeared in the former sections, a "ball" refers to a white spherical object inside the simulation. This is the object that robots in the simulations try to return for fulfilling rallies, exchanges of the ball between two players. We manually programmed the rules of air drag applied to a shuttlecock so that the trajectory of a ball resembles that of a shuttlecock in the physical world, where a shuttlecock hit by a player starts flying at a very high speed but soon decelerates and reaches the other player with relatively modest speed. Pseudo-code for dynamic alteration of the ball's speed is provided in Algorithm 1. At the beginning of each episode, the ball is shot toward a player with randomized velocity and starting position.

F. Task Explanation

To ultimately succeed in the rally task explained in III-A, two different tasks are designed to train agents progressively.

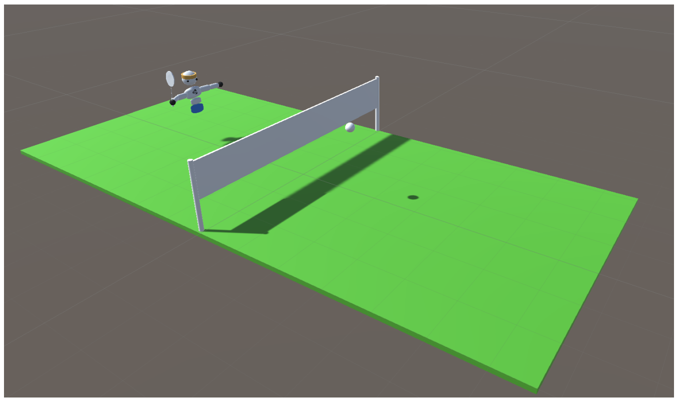
Fig. 3. A snapshot of the simulation environment with a human model (HUMAN-LIKE).

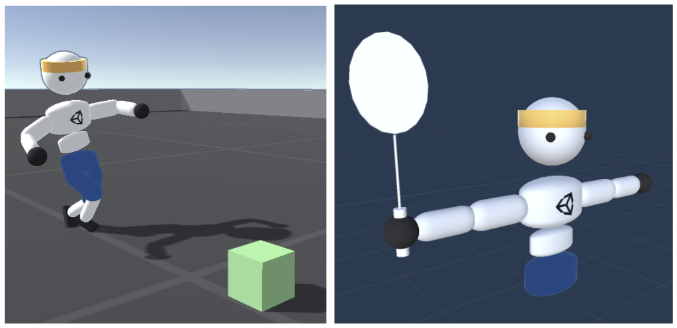
Fig. 4. The default Walker agent from ML-Agent Toolkit(left), and our player agent created from the Walker(right).

Algorithm 1 Dynamic alteration of the ball's speed

```
INPUT: Initial speed of the ball $i$ OUTPUT: Speed of the ball at each frame $s$
  Initialize flying duration $f$,
  $s \leftarrow i$
  for each frame do
      $f \leftarrow f$ + SECONDS-PER-FRAME
      if $f < 0.6$ then
          $s \leftarrow -f + i$
      end if
      Return $s$
  end for
```

We call the tasks the Low-, a High-level tasks, respectively.

An agent is first trained for the Low-level task. The purpose of the agent in the Low-level task is to be able to return balls into the opponent's court in diverse ways. There is only one player in the task. The player is placed on one side of the court, and a ball is shot toward the player. The player tries to return the ball by changing the position to reach it and rotate its body joints to realize a hitting motion. The reward functions are different in SIMPLE and HUMAN-LIKE. In SIMPLE, a target position that indicates where the player aims to return the ball is sampled at the beginning of each episode. When the player successfully hits the ball, and it lands on the opponent's

TABLE III
List of Joints with Constraints

Joint Name	X-Axis	Y-Axis	Z-Axis
Spine	*Locked*	*Locked*	-15 ∼ 15
Chest	*Locked*	*Locked*	-15 ∼ 15
Right Upper Arm	-90 ∼ 90	-75 ∼ 75	-85 ∼ 85
Right Lower Arm	0 ∼ 100	*Locked*	*Locked*
Right Hand	-90 ∼ 90	-25 ∼ 25	0

TABLE IV
Elements Added to the State Space Per Joint in HUMAN-LIKE

Property Name	Vector Size
Velocity	3
Angular Velocity	3
Position	3
Orientation	4
Magnitude of Applied Force	1
Total	14

court, we calculate the episode reward as:

$$R = R_{\text{hit}} + R_{\text{dist}}. \tag{1}$$

R_{hit} takes a value of either 0 or 1, depending on whether the player hits the ball. When the ball hit by the player lands on the opponent's court, R_{dist} is calculated as:

$$R_{\text{dist}}(d) = \text{Exp}(-0.5d) \tag{2}$$

where the argument d is a distance between the target position and the position where the ball landed.

In HUMAN-LIKE, we use a more explicit target value than that in SIMPLE. At the beginning of each episode, a target velocity, which indicates the velocity at which a player is requested to hit the ball with its end effector, is sampled. The reward is calculated when the player hits the ball, following the equation:

$$R = 0.5(R_{\text{hit}} + R_{\text{velocity}}). \tag{3}$$

R_{hit} is the same term as that in SIMPLE. R_{velocity} is computed following the formula:

$$R_{\text{velocity}}(d) = \text{Exp}(-1.5d) \tag{4}$$

where d is the L2 distance between the target velocity and the actual velocity when the racket collides with the ball, respectively.

In tasks in both simulations, the ball is fed from a randomized position and with a varied velocity at the beginning of each episode. An episode is terminated when the ball lands on a court surface, regardless of whether the player hits it.

Another task is the High-level task, which is implemented only in SIMPLE. In this task, two players are supposed to have rallies. Each of them is placed on either side of the court facing one another. Once the ball is fed to either of them when an episode starts, they return the ball to their opponent's court in turns. We note that since the player needs to consider where their opponent is when deciding where to hit the ball, the position of the opponent is incorporated into the action space of the task environment in addition to the properties described in Table. I.

TABLE V
State Space of HUMAN-LIKE

Property Name	Vector Size
Spine	14
Chest	14
Right Upper Arm	14
Right Lower Arm	14
Right Hand	14
Position of the Ball	3
Velocity of the Ball	3
Position of the Player	3
Position of the Racket Head	3
Target Velocity	3
Total	85

TABLE VI
Action Space of HUMAN-LIKE

Property Name	Vector Size
Value of Movement	2
Target Rotation of "Spine"	1
Target Rotation of "Chest"	1
Target Rotation of "Right Upper Arm"	3
Target Rotation of "Right Lower Arm"	1
Target Rotation of "Right Hand"	2
Force Applied to Each Joint	5
Total	15

We set the task to a cooperative setting in this research. An episode reward gets a positive value of 0.2 each time one of the players hits the ball.

IV. Training Procedure

To train a model that can play a rally, we divide the training process into two parts, low-level and high-level training. Each of them is executed in the Low-level and High-level task in the simulation, respectively.

In this research, we use goal-conditioned HRL. In HRL architecture, there are two different policies instead of only one, as in most of the other RL algorithms. We call the first policy a low-level policy formulated as $\pi_\theta^{low}(a|s,g)$ where a and s indicate action and observation of the task, and g represents the goal value explained in the earlier section. In the Low-level task in SIMPLE, a trained low-level policy is supposed to output proper arm movement to return a ball to the position described by the value of g.

The other policy is called a high-level policy defined as $\pi_\theta^{high}(g|s)$. The high-level policy is responsible for determining the target value g based on the given input observation s, which means to decide where in the opponent's court the player returns the shuttle.

The two policies introduced above function as one inference model to output the ultimate movement of the player. When any state is given, the high-level policy takes it as input to return a goal, which is given to the low-level policy with the input state to output its action.

As shown in Figure 5, we start by training a low-level policy $\pi_\theta^{low}(a|s,g)$ in the low-level task described in Chapter 4. Once we have the trained $\pi_\theta^{low}(a|s,g)$, we load it into the high-level task. This way, players in the environment can return balls

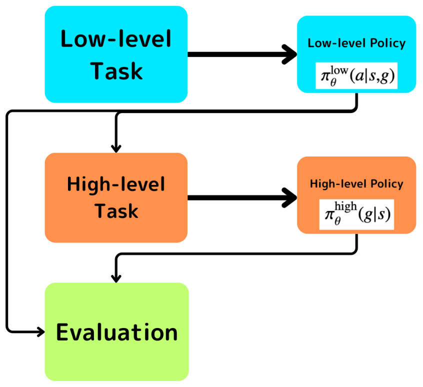

Fig. 5. The Training Pipeline.

from the beginning of the high-level training, so we only focus on updating the high-level policy. We only train one policy at a time, so when one of the policies is modified, the other is frozen throughout the training phase.

We aim to employ the HRL approach because we theorize that its two-stage inference flow mirrors human decision-making in sports. Human players first decide on their objective during games, and think about how to execute it by moving their bodies accordingly. By modeling this cognitive process with an HRL framework, we can imbue the agent with an inductive bias, potentially enhancing sample efficiency of the training. The training code is provided in https://github.com/shuny42657/RallySim

V. Results

We conduct some experiments to validate the efficacy of HRL for the rally task. The results are discussed in this chapter.

We compare the quantitative results of the model trained by our suggested method with the one trained through a single-policy approach. We also spectate the rendered results of the models performing each task. From the obtained results we conclude that our hierarchical approach significantly improves the overall performance of a player in the badminton tasks and the learning efficiency.

A. Training Results of SIMPLE

We first evaluate the quantitative and qualitative results in Low-level and High-level tasks. We use the PPO algorithm throughout the training [18], [19].

Fig. 6 illustrates the reward in the low-level task conducted in SIMPLE with 2,000,000 training steps. Values are plotted at the evaluation phase that comes every 5000 training steps during the training.

The reward value reached around 1.8. We also rendered the trained model to evaluate its performance quantitatively. Fig. 7 shows snapshots of an episode where the robot aims to return the ball to the place shown with the red sphere. The robot

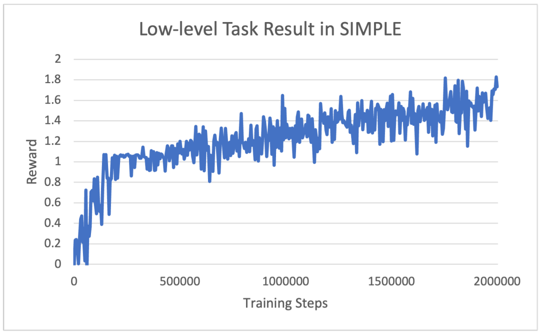

Fig. 6. Reward in the low-level task in SIMPLE, with 2.0 being the maximum reward.

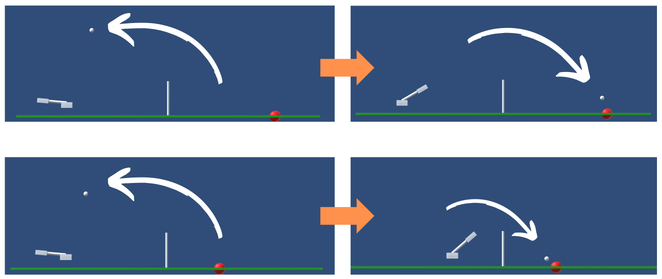

Fig. 7. Rendered results of a trained agent successfully returning the ball to the target positions.

successfully moved to a good position for it to hit the ball, able to return it to the place close enough to the target position.

For the High-level task, we prepared two methods to compare with our proposed methods(Ours). We call them “Random High-level” and “Monolithic,” respectively. Random High-level only trains the low-level agent and uses random high-level action(goals) when playing rally in the high-level task. Monolithic is named after its architecture, which only has a single-policy network, as most other RL algorithms. It skips the training of the low-level task and directly trains a model with the PPO algorithm in the high-level task. Table VII shows the evaluation results of each method with how many times players trained by each method could exchange the ball (rally length). We compared both the average rally length and the number of times a rally reached the maximum rally length of 50. Ours surpasses the other two methods in both categories. The fact that players trained by Ours play longer rallies than Monolithic validates our hypothesis that the rally problem is a complicated task that cannot be solved well by using a single-model policy, but HRL can effectively tackle it. Ours also outperformed Random High-level. Supported by this fact, we also argue that only having a trained low-level policy is not enough to solve the rally task, but learning a strategic factor of the task in a high-level training phase plays an important role in displaying a desirable performance in it.

We observed the rendered results of the high-level tasks as we did so in the low-level task. Players trained by Ours tend to maintain a stable rally by choosing to return the ball to the middle of the opponent’s court, which is thought to be an

TABLE VII
EVALUATION RESULTS OF EACH METHOD IN THE HIGH-LEVEL TASK

Method	Average Rally Length	Number of Rallies with Length of 50
Ours	20.25	13
Random High-level	12.90	3
Monolithic	1.32	0

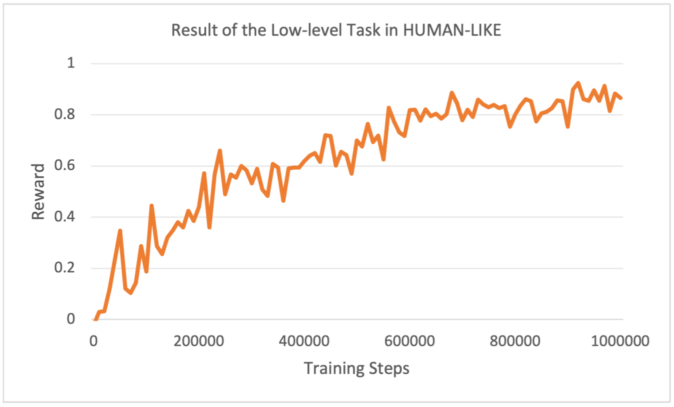

Fig. 8. The training result of the Low-level task in Human-Like

easy kind of shot to return since the opponent does not need to move a lot to reach the ball compared to the situation in which it falls onto the front side of the court, or it flies to the rear side. This basic principle in the players' choice of shots makes the rally stable unless one of them sometimes misses or fails to return the ball to the target position set in the middle of the opponent's court.

Players trained by Random High-level can return shuttles to their opponent's court, but since cannot choose a target place, the ball often flies to spots where it is hard for their opponent to return it successfully. The rally is more likely to get unstable than that played by players trained by ours, making the average rally length shorter. Monolithic models can maintain the rally for few times, but the players cannot to cope with all the shots that land in different places on the court. Nor can they return the ball in the way their opponent can easily hit it. The evaluation code for SIMPLE environment is provided in https://github.com/shuny42657/RallySimEvaluation

B. Training Results of HUMAN-LIKE

In this research, we only conducted the low-level task in HUMAN-LIKE. The reason why we did not test our low-level model trained in HUMAN-LIKE in the high-level task is that the trained model did not display the required performance needed for it to be used in the high-level task. Since HUMAN-LIKE has 3 dimensions the trajectories of the ball that a player needs to hit get drastically diverse compared to those in SIMPLE; it was hard for the model to learn to cope with all the situations. The model we currently have after training can return the ball that follows limited kinds of trajectories.

Figure 8 shows the result of the low-level training of our proposed method(Ours) in HUMAN-LIKE.

Although the learning curve did grow to an extent, when we rendered the simulation of the Low-level task in HUMAN-LIKE the result did not exhibit satisfactory outputs. Although the player diversifies the ball's behavior to some extent, comparing the velocity of the racket when it collides with the ball with the target velocity, they are different at least by approximately 10 degrees in Euler angles. It is, therefore, not capable of returning the ball to the extent we intuitively feel that its control is accurate enough.

The player in HUMAN-LIKE is designed to be able to perform movements that resemble that of humans. Further research is expected to create a method that can make the player act in a more sophisticated way.

VI. CONCLUSION

We proposed RallySim as a novel cooperative multi-agent problem that involves both motor control and acquisition of strategy. HRL displays prominence in training an agent that can successfully perform a long rally in the task. Training in the 3D environment leaves room for further exploration.

ACKNOWLEDGMENT

This work was partially supported by JST Moonshot R&D Grant Number JPMJPS2011, CREST Grant Number JP-MJCR2015 and Basic Research Grant (Super AI) of Institute for AI and Beyond of the University of Tokyo.

REFERENCES

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.
[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.
[3] Smruti Amarjyoti. Deep reinforcement learning for robotic manipulation-the state of the art, 2017.
[4] Saminda Abeyruwan, Laura Graesser, David B. D'Ambrosio, Avi Singh, Anish Shankar, Alex Bewley, Deepali Jain, Krzysztof Choromanski, and Pannag R. Sanketi. i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops, 2022.
[5] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation, 2021.
[6] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018.
[7] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2021.
[8] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020.
[9] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning, 2018.
[10] Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning?, 2019.
[11] Kandai Watanabe, Mathew Strong, and Omer Eldar. Shiro: Soft hierarchical reinforcement learning, 2022.
[12] Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019.

[14] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents, 2020.

[15] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. 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, 2019.

[16] Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, S. M. Ali Eslami, Daniel Hennes, Wojciech M. Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, Noah Y. Siegel, Leonard Hasenclever, Luke Marris, Saran Tunyasuvunakool, H. Francis Song, Markus Wulfmeier, Paul Muller, Tuomas Haarnoja, Brendan D. Tracey, Karl Tuyls, Thore Graepel, and Nicolas Heess. From motor control to team play in simulated humanoid football, 2021.

[17] Şahin Mercan and Pinar Onay Durdu. Evaluating the usability of unity game engine from developers' perspective. *2017 IEEE 11th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–5, 2017.

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[19] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022.