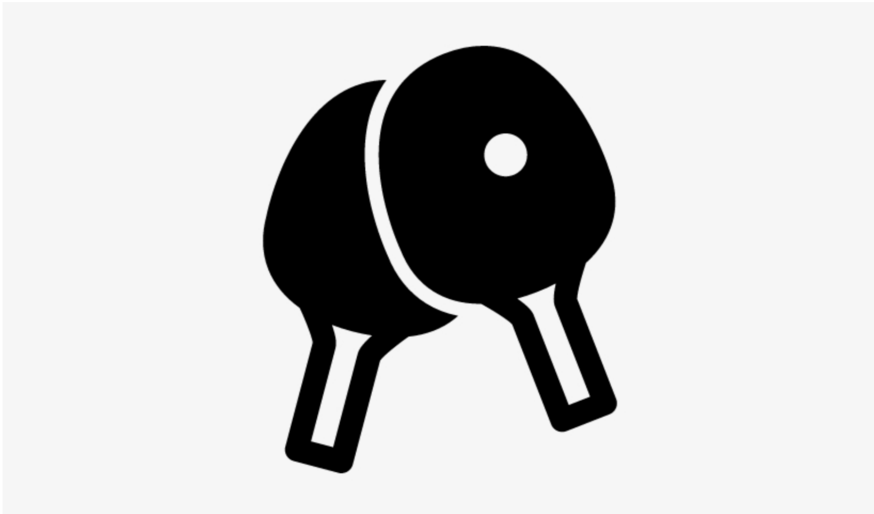


CSCI 527

Applied Machine Learning for Games

Engineering Design Document

Version 1.0

Authors

Aratrika Basu	aratrika@usc.edu
Jayantraj Coimbatore Selvakumar	jcoimbat@usc.edu
Justin Fu	ziyuefu@usc.edu
Manikanta Chunduru Balaji	mchundur@usc.edu
Shruthi Ramesh	shruthir@usc.edu
Sowmya Voona	voona@usc.edu
Yue Wu	ywu73061@usc.edu

October 19, 2021

Contents

1 Introduction

1.1 Purpose and Goal

This document provide an overview of *Table Tennis Agent* application, developed by Team Malong for the course CSCI 527:Applied Machine Learning for Games. The objective of our project is to understand how we can use machine learning algorithms to train dual AI agents to play a game of table tennis. Through this process we would like to develop an optimized algorithm that will enable each agent to play table tennis as per the rules. For this purpose we believe to use an actor critic reinforcement learning approach in order to train each agent to maximize its performance. For further research, we want to explore the realm of multi-agent learning, developing fully competitive multi-agent RL methods that could well perform in table tennis.

We want to achieve the following goals.

1. Build a table tennis environment in Unity with the following engineering requirements.
   (a) Neat graphics and visual effects of a 3D table tennis game.
   (b) Both human and AI are available to play the game.
   (c) have feed-backs like scoreboard, showing winner or loser, etc
2. Build table tennis agent that could outperform human and gain highly competitive performance.
   (a) Fully utilize the unity-ml toolkit and train with popular RL methods like PPO, SAC, DQN
   (b) Research on multi-agent RL methods.

The following is a schedule for our project development.

Milestone	Expected Completion Date
Study reinforcement learning basics	Week 1-5
Research unity ml-agent	Week 4-6
Research self defined models	Week 6-7
Train basic RL agent using PPO, SAC, etc	Week 7-9
Improve RL agent with other ml-agent models(DQN etc)	Week 9-11
Deploy MARL methods to game	Week 11-13
Train MARl agents and compare it with RL agents	week 14-15

1.2 Background and Overview

1.3 Background

Table Tennis, known as Ping-Pong, is a popular sport in which two or four players hit a lightweight ball on a hard table divided by a net[1]. There has been many attempts to use robots to play table tennis in real-world. But many of those research focus more on perception, such as tracing the trajectories of the ball and predicting its position, transferring the learned agents in simulation environment to real-world setting. They could not compete against human players at any level, either dynamically or strategically. On the other hand, there are also plenty of table tennis games available on virtual environment, like PC, console and mobile devices. However, they are mostly platforms for two human players play against each other, while the provided AI are mostly rule-based. Our table tennis agents aim to achieve high performance under virtual environment, so that learning the strategies are more important to us compared with many prior works on table tennis robots. As a result, we want our environment to be not only neat and clean, but also to consider physical properties like spinning and collision coefficients.

1.4 Design Overview

TODO A system overview graph. Reference: `https://robotics.ee.uwa.edu.au/courses/design/examples/example_design.pdf` Page 22. Also some brief

2 Prior Works

2.1 Reinforcement Learning

Reinforcement learning is a type of machine learning that uses AI systems to follow a policy in order to learn an objective and there by maximize the cumulative reward [2]. Here the AI system starts learning step by step by trial and error approach. For every correct action it performs it is given a reward and for any subsequent mistake it receives a penalty. Using this feedback mechanism of reward and penalty reinforcement learning learns well in the environment around it[3]. With the development of deep learning, neural networks empower RL with unprecedented abilities in field of Go[4], Atari[5], StarCraft[6], Robotics[7]. A major family of RL algorithms are policy optimization, where they represent a policy as $\pi_\theta(a\|s)$. The parameters θ are optimized by gradient decent of objectives, often involving learning value functions at the same time. Some representative algorithms are actor-critic algorithm[8], which is temporal difference version of policy gradient, A2C[9], which directly performs gradient ascent in asynchronous manner, and PPO[10], which indirectly maximize a surrogate objective function. Another Family is based no action value function, called Q-Learning, first raised by Watkins in 1992[11].

2.2 Table Tennis Robots

Attempts to use robots to play table tennis could be traced back to the 80s. Since Anderson[12] built a real-world vision system which subjectively evaluates and improves its motion plan as the data arrives, many table tennis robot systems were built[13], [14],[15],[16],[17],[18].

As the development of deep learning, especially reinforcement learning, training a robot to play table tennis in real world has been made possible. Lately, Wenbo et al.[19] demonstrate a model-free approach mixed of evolutionary search and CNN-based policy architectures. Jonas et al.[20] shows a modified DDPG[21] could increase sample efficiency in table tennis. Büchler et al., combines step-based reinforcement learning with pneumatic artificial muscles, and achieved great performance using a hybrid sim and real training process. For further learning, Matsushima summarizes the many learning approaches in robotic table tennis[22].

2.3 multi-Agent learning

Generally, in gaming, it often involves the participation of more than one single agent, which fall into the real of multi-agent RL(MARL). As the previous papers mostly try to solve table-tennis training a single agent, we want to capture the competitive nature of a sport, thus training a pair of agent against each other. MARL algorithms are widely known to be sample-inefficient and millions of interactions are needed. For the game of table tennis, the interaction between the agent and the environment is relatively simple compared with games like starcraft. Hence, we would focus more on the model-free setting, where the policies are learned without direct access to the environment.

Compared with single-agent RL, MARL suffers from several challenges. As summarized by [23], MARL does not have unique learning gols and whether convergence of equilibrium point is the alpha performance criterion for MARL algorithm analysis is controversial. Some researchers found value-based MARL algorithms fail to converge to stationary Nash equilibrium point for general-sum Markov games [24]. Another major issue is the non-stationary setting as multiple agents could simultaneously interact with the environment and each other. This could bring challenge to value estimation as well as policy optimization during training. Scalability is a issue comming along with non-stationary, as the joint action space is exponentially increasing. Even in a dual agent setting as table tennis, the sample efficiency would still be a major bottleneck.

MARL has many information structures(who knows what at the trainign and execution) [23]. For the dual agent setting of table tennis, the straight forward way is treat other agent as part of the environment, which is called *Independent Learning*(IL). But IL face the problem of non-stationary dynamic, which harms the performance of policies. Some work try to stablize the learning process[25], [26]. Others try to build communication protocals between agents [27],[28]. Another major MARL learning diagram is *Centralized Training and Decentralized Execution* (CTDE). One represetative CTDE method is MADDPG[29], a multi-agent version actor-critic. Each agent maintains its own critic Q_i, which estimates the joint value function and uses the critic to update its decentralized policy.

MARL consists of three groups, fully cooperative, fully competitive and mixed of two. Though Table tennis is considered to be competitive game, we would also try some mixed methods since table-tennis is not a typical *zero-sum* game, where the reward for one player is exactly the loss of the other.

3 Methods

3.1 Preliminaries

The Table Tennis Game could be described as a Markov Process, and is a Markov Game[30].

Markov Decision Process(MDP). An MDP is defined as

$$< S, A, T, R, \rho, \lambda >$$

where S a set of states, A a set of actions, $T : S \times A \to P(S)$ a stochastic transition function, $R : S \times A \to R$ a reward function, $\lambda \in [0, 1)$ a discount factor. The agent(table tennis player) interacts with the ball by performing its policy $\pi : S \to P(A)$. The agents learn this policy to maximize the expected cumulative discounted reward:

$$J(\pi) = E_{\rho,\pi,T} \sum_{t=0}^{\infty} r_t \lambda^t$$

where $r_t = R(s_t, a_t)$, $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(s_t)$, $s_{t+1} \sim T(|s_t, a_t)$

Markov Game(MG). An MG is an extension of MDP and is defined as

$$< S, N, \{A^i\}_{i=1}^N, \{R^i\}_{i=1}^N, \{O^i\}_{i=1}^N, \rho, \lambda, Z >$$

where the action sets now contain N agents, namely, $A^1 \cdots A^N$, state transition function $T : S \times A^1 \cdots A^N \to P(S)$, reward function $R : S \times A^1 \cdots A^N \to R$. For partially observable Markov games, each agent i receives local observation $o^i : Z(S, i) \to O^i$ and interacts with environment with its policy $\pi^i : O^i \to P(A^i)$. The expected cumulative discount reward now turns into

$$J^i(\pi^i) = E_{\rho,\pi^1,\cdots,\pi^N,T} \sum_{t=0}^{\infty} {r^i}_t \lambda^t$$

where ${r^i}_t = R^i(s_t, a_t^1, \cdots, a_t^N)$. Recently, Reinforcement learning has become efficient in solving Markov Games, we would discuss methods like PPO, SAC and DQN in later sections.

3.2 Table Tennis Environment

Software Descriptions. TODO Reference: `https://robotics.ee.uwa.edu.au/courses/design/examples/example_design.pdf` Page 25.

This sections describes in detail about how the environment is setup for the project.

3.2.1 Development Environment

1. **Unity 3D: 2020.3.20** Unity is a cross-platform game engine developed by Unity Technologies. The game engine can be used to develop interactive 3D, 2D, as well as interactive simulations and other experiences[31]. Unity version 2020.3.20 is utilized for the environment setup.

2. **Unity Machine Learning Agents Toolkit**

   The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents. They provide state-of-the-art algorithms which can be used to train intelligent agents to play different 3D and 2D games. The ML agents package provides an option to convert a Unity scene into a learning environment where character behaviors can be trained using machine learning algorithms.

3. **Pytorch** PyTorch is an open-source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing[32].

4. **Python**

5. **Tensorboard** It is a Tensorflow visualization toolkit that provides visualization and tools for machine learning experimentation. It helps to track and visualizing metrics like loss and accuracy. Tensorflow.dev provides an easy way to share ML experimentation results.

6. software overview, a figure of the code structure(could check the reference above)

7. briefly explain the software design overview.

8. Progress summary for building the environment, a table. Similar to section 1.3 Goal.

Milestone	ate
xxx	Week x

3.2.2 Methodology

Our project aimed to create an environment and ML agents, to enable them to play a game of table tennis. For this purpose, we found out that Unity provides a comprehensive environment where game objects can be created and modeled as per user requirements. Also, game objects can be used as ML agents and can be trained using Proximal Policy Optimization and Soft Actor-Critic model provided by Unity. So we selected the Unity platform as the environment.

Our environment contains a Table Tennis bat with the ability to assign different unity materials to different sections of the bat for customizability. The sections include Bat forehand face, Bat backhand face, Bat center, and the Bat handle. The second model in this pack is the table tennis table which can have different unity materials assigned to it for customizability. The sections include Tabletop, Table Legs, Net frame.

1. **Vector Observations:** From the environment, we are collecting the positions of bat A, bat B, and the ball. Also, we are collecting the velocity of bat A, bat B, and the ball. We have used these observations to train the model using different Reinforcement learning algorithms.

2. **Actions:** We have designed the environment in a way where the bats can move along X and Y axes and can rotate along X axes. Our goal is also to use the Z axes in the following weeks. The bats can also be moved using the 'right', 'left' keys.

3. **Reward Policy:** We have designed our reward policy in such a way that if a player commits a mistake or makes a foul move the opponent player gets the reward for it. The following are the foul moves implemented for our project:

    - Player hitting the ball to the net.
    - Player hitting the ball over the boundary.
    - Ball bouncing more than once on the same side of the court.

    For implementing the reward policy we are keeping track of the parameters given below:

    - Last Hit Agent : The agent who hit the ball previously before coming to the current player.
    - Last Collided With : This keeps track of the last surface the ball collided with. Here the surface refers to the court of player A, court of player B, Net, etc.
    - Next Agent Turn: This keeps the track of the next agent who has to hit the ball to continue the game.

    Using the above parameters we are rewarding the agents.

3.2.3 Software Overview:

The Software Overview consists of four classes:

1. Game Controller Class: This is the main controller class that is interlinked to all other classes. It has the following functionalities:

- agentScores(): This method is used for rewarding the agents.
- episodeReset(): It is used to reset the episode for every foul move.
- matchReset(): It is used to reset the match.
- ballHitsAgent(), ballHitsFloor(), ballHitsBoundary(), agentHitsNet(), ballHitReward(): These methods are used to handle the reward for the agent depending on the foul moves described for the game.

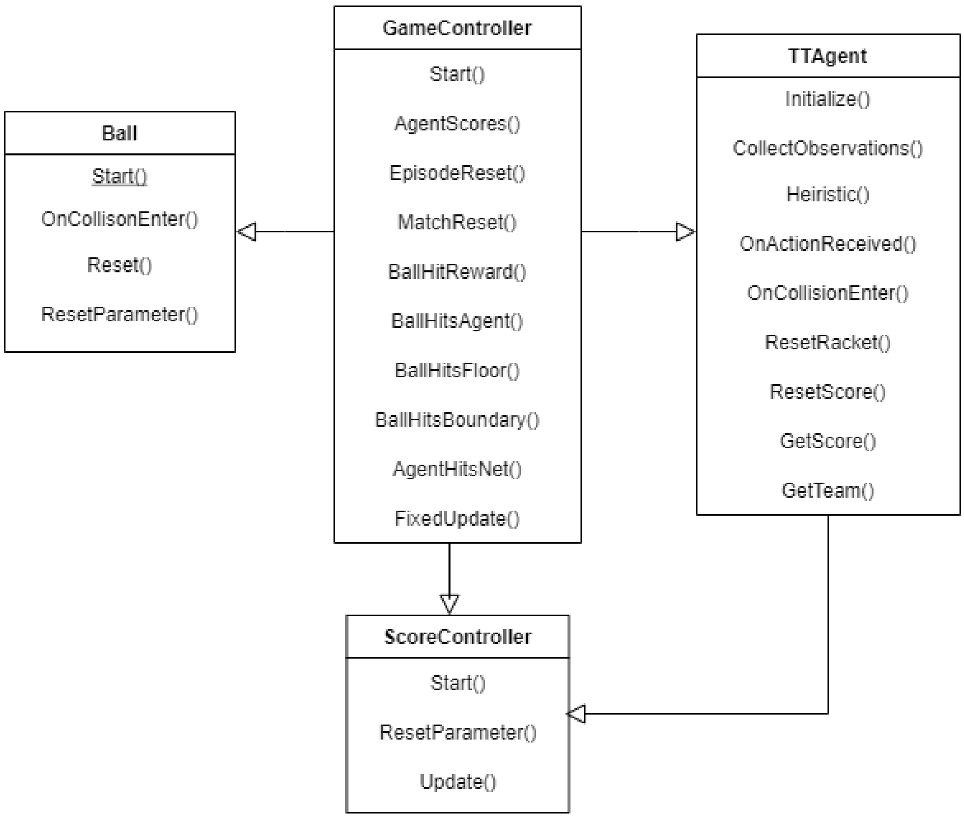


Figure 1: Software Workflow Diagram

2. Ball Class: This class refers to the functionalities used for the ball.
   - onCollisionEnter(): This function handles the different nuances when the ball collides with different surfaces. For example, when the ball collides with bat A we are checking the parameter status of the lastHitAgent, lastCollidedWith and we are rewarding the agent as per the rules of table tennis.
   - reset(),resetParameter(): This functionality handles resetting the ball positions for every episode.

3. TTAgent Class: This class involves all the functionalities required for the agent.

   - CollectObservations(): It is used for collecting the vector observations for the bats and the ball. The velocity observations of the bats and the ball are also collected.
   - Heuristic(): This functionality is used to assign the movement to the bat along 'x' and 'y' axis. The bat can be moved along the horizontal axis using the right and left keys. It can be moved in the vertical direction using the upward and downward keys. The racket can be made to jump using the 'X' key.
   - OnActionReceived: This executes the actions by moving the game objects in the vector space.
   - resetRacket: It is used to reset the racket position.
   - resetScore: It is used to reset the score for the agents.

4. Score Controller Class:

3.3 Modeling and Training

3.3.1 PPO

1. Model description: Proximal Policy Optimization(PPO) is an on-policy based reinforcement learning algorithm. This algorithm was introduced by the OpenAI team in the year 2017 [10] and quickly became one of the most popular RL methods surpassing the Deep-Q learning method. PPO is scalable, data efficient, and successful on a variety of problems without hyperparameter tuning.

   PPO is an algorithm that attains the data efficiency and reliable performance of trust region policy optimization (TRPO), while using only first-order optimization.It involves collecting a small batch of experiences interacting with the environment and using that batch to update its decision-making policy. Once the policy is updated with this batch, the experiences are thrown away and a newer batch is collected with the newly updated policy. This is the reason it is an "on-policy learning" approach where the experience samples collected are only useful for updating the current policy once.

   PPO improves stability of the learning by mainly 2 techniques:

   - Clipped Surrogate Objective: The Clipped Surrogate Objective is a drop-in replacement for the policy gradient objective that is designed to improve training stability by limiting the change you make to your policy at each step.

- Multiple epochs for policy updating : Unlike vanilla policy gradient methods, and because of the Clipped Surrogate Objective function, PPO allows user to run multiple epochs of gradient ascent on your samples without causing destructively large policy updates. This allows to squeeze more out of your data and reduce sample inefficiency.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
    for actor=$1, 2, \ldots, N$ **do**
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2: Proximal Policy Optimization Algorithm

2. Training:

   (a) The training is carried out by setting the behavior type of agents to "Default" in Unity so that no external/human interaction is required to play the game. We used the mlagents-learn package to execute the configuration file which contains the hyper parameters specific to each model. Each time a configuration file is called a new model is trained and gets saved in the local system. Later, the trained model can be embedded into Unity as the model type in order to observe the learning that the agents have obtained.

   We have tuned the model by using a variety of hyper parameter combination in our configuration file while keeping our batch size as 2048, hidden units in each layer as 256 and initial ELO as 1200. The table [1] contains the hyper parameter combinations.

3.3.2 SAC

1. Model description: Soft Actor Critic(SAC) is an off-policy model-free reinforcement learning algorithm. This RL algorithm was developed jointly by UC Berkely and Google and was introduced in the year 2018 [10]. It is considered one of the most efficient algorithm to be used in real-world robotics.

Buffer Size	Learn Rt	Epochs	Learn Sch	Layers	Max Steps	Final ELO
2048000	0.0003	3	constant	3	370000	1202
2048000	0.0003	3	constant	2	1.6M	1191
2048000	0.001	3	constant	2	1.93M	1208
20480	0.03	3	constant	2	730000	1203
20480	0.01	3	constant	2	2.23M	1190
20480	0.01	3	constant	3	2.19M	1170
20480	0.01	500	linear	3	20000	1193
20480	0.01	10	linear	3	1.2M	1189
20480	0.01	1000	linear	3	100000	1195
20480	0.01	100	linear	3	1M	1205

Table 1: PPO Hyper parameter Combination

The biggest feature of SAC is that it uses a modified RL objective function. Instead of only seeking to maximize the lifetime rewards, SAC seeks to also maximize the entropy of the policy. A high entropy in our policy explicitly encourages exploration, encourages the policy to assign equal probabilities to actions that have same or nearly equal Q-values, and also ensures that it does not collapse into repeatedly selecting a particular action that could exploit some inconsistency in the approximated Q function. SAC overcomes the brittleness problem by encouraging the policy network to explore and not assign a very high probability to any one part of the range of actions.

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Figure 3: Soft Actor-Critic Algorithm

2. Training:

   (a) The training is carried out by setting the behavior type of agents

Buffer size	Batch size	Learn Rate	Buffer init steps	Layer	Steps	Final ELO
500000	128	0.0003	0	2	7.5M	2352
50000	128	0.01	0	2	2M	1272
500000	128	0.003	0	2	3.6M	1540
1000000	1024	0.0003	1000	2	4M	2002
500000	512	0.0003	1000	2	3M	1953
500000	512	0.0003	1000	2	10M	2130
1000000	1024	0.0003	1000	3	3.9M	1915
1000000	1024	0.0003	1000	3	0.7M	1748
500000	512	0.003	0	3	1.98M	2005
500000	512	0.0003	0	2	Prog.	Prog.

Table 2: SAC Hyper parameter Combination

to "Default" in Unity so that no external/human interaction is required to play the game. We used the mlagents-learn package to execute the configuration file which contains the hyper parameters specific to each model. Each time a configuration file is called a new model is trained and gets saved in the local system. Later, the trained model can be embedded into Unity as the model type in order to observe the learning that the agents have obtained.

We have tuned the model by using a variety of hyper parameter combination in our configuration file while keeping our hidden units in each layer as 256, learning rate schedule as 'constant' and initial ELO as 1200. The table [2] contains the hyper parameter combinations.

3.3.3 DQN

1. Model Description : DQN is an off-policy, value-based, model-free RL algorithm. This algorithm was introduced by DeepMind Technologies in the year 2013 [33]. The algorithm was modified in the 2015.

   A DQN, or Deep Q-Network, approximates a state-value function in a Q-Learning framework with a neural network. In the Atari Games case, they take in several frames of the game as an input and output state values for each action as an output.

   It is usually used in conjunction with Experience Replay, for storing the episode steps in memory for off-policy learning, where samples are drawn from the replay memory at random. Additionally, the Q-Network is usually optimized towards a frozen target network that is periodically updated with the latest weights every steps. The latter makes training more stable by preventing short-term oscillations from a moving target. The former tackles autocorrelation that would occur from on-line learning, and having a replay memory makes the problem more like a supervised learning problem.

DQN overcomes unstable learning by mainly 2 techniques.

- Experience Replay
- Target Network

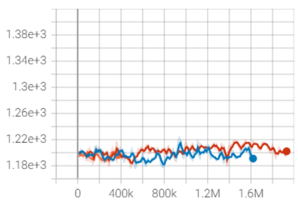
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

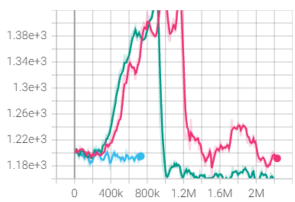
Figure 4: DQN Algorithm

TODO: SAC graphs once training completes

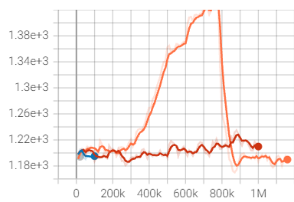
3.4 Observation



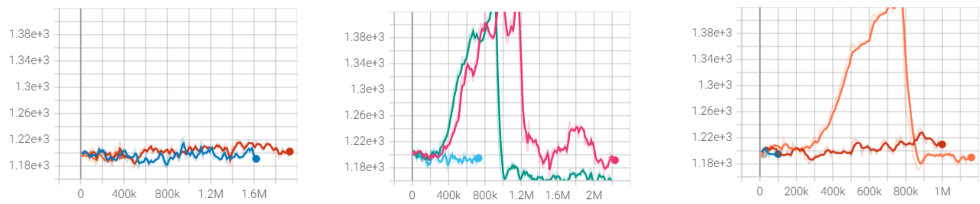
(a)



(b)



(c)



(d) (e) (f)

Figure 5: ***PPO training Observation****: Self Play/ELO graphs with the hyper parameter combination:(a) High Buffer Size of 2048000, low Learning Rate of 0.0003, (b) Low Buffer Size of 20480, high Learning Rate of 0.01, epochs of 3, (c) Low Buffer Size of 20480, high Learning Rate of 0.01, epochs of 500, 10, 1000. 100.* ***SAC training Observation****: Self Play/ELO graphs with the hyper parameter combination:(d) High Buffer Size of 2048000, low Learning Rate of 0.0003, (e) Low Buffer Size of 20480, high Learning Rate of 0.01, epochs of 3, (f) Low Buffer Size of 20480, high Learning Rate of 0.01, epochs of 500, 10, 1000. 100*

References

[1] Wikipedia. Table tennis — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Table%20tennis&oldid=1039724889, 2021. [Online; accessed 06-September-2021].

[2] Wikipedia contributors. Reinforcement learning — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1042314112, 2021. [Online; accessed 6-September-2021].

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[4] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[5] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. 2017. cite arxiv:1710.02298Comment: Under review as a conference paper at AAAI 2018.

[6] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019.

[7] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[8] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[11] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Mach. Learn.*, 8:279–292, 1992.

[12] Russell L. Anderson. *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control.* MIT Press, Cambridge, MA, USA, 1988.

[13] Fumio Miyazaki, Michiya Matsushima, and Masahiro Takeuchi. Learning to dynamically manipulate: A table tennis robot controls a ball and rallies with a human being. *Advances in Robot Control: From Everyday Physics to Human-Like Movements*, 01 2006.

[14] Katharina Mülling and Jan Peters. *A Computational Model of Human Table Tennis for Robot Application*, page 57. 2009.

[15] Katharina Muelling, Jens Kober, and Jan Peters. Learning table tennis with a mixture of motor primitives. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 411–416, 2010.

[16] Yanlong Huang, Dieter Buchler, Okan Koc, Bernhard Schölkopf, and Jan Peters. Jointly learning trajectory generation and hitting point prediction in robot table tennis. In *16th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2016, Cancun, Mexico, November 15-17, 2016*, pages 650–655. IEEE, 2016.

[17] Reza Mahjourian, Navdeep Jaitly, Nevena Lazic, Sergey Levine, and Risto Miikkulainen. Hierarchical policy design for sample-efficient learning of robot table tennis through self-play. *CoRR*, abs/1811.12927, 2018.

[18] Katharina Muelling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. In *AAAI Fall Symposium on Robots that Learn Interactively from Human Teachers*, pages 263–279, 2012.

[19] Wenbo Gao, Laura Graesser, Krzysztof Choromanski, Xingyou Song, Nevena Lazic, Pannag Sanketi, Vikas Sindhwani, and Navdeep Jaitly. Robotic table tennis with model-free reinforcement learning, 2020.

[20] Jonas Tebbe, Lukas Krauch, Yapeng Gao, and Andreas Zell. Sample-efficient reinforcement learning in robotic table tennis, 2021.

[21] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[22] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki. A learning approach to robotic table tennis. *IEEE Transactions on Robotics*, 21(4):767–771, 2005.

[23] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2021.

[24] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. Technical report, 2003.

[25] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient multi-agent deep reinforcement learning. *CoRR*, abs/1707.04402, 2017.

[26] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation, 2016.

[27] Xiangyu Kong, Bo Xin, Fangchen Liu, and Yizhou Wang. Revisiting the master-slave architecture in multi-agent deep reinforcement learning, 2017.

[28] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NIPS*, pages 2137–2145, 2016.

[29] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *NIPS*, pages 6379–6390, 2017.

[30] Michael L. Littman. Value-function reinforcement learning in markov games. *Cogn. Syst. Res.*, 2(1):55–66, 2001.

[31] Wikipedia contributors. Unity (game engine) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1049877901, 2021. [Online; accessed 16-October-2021].

[32] Wikipedia contributors. Pytorch — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=PyTorch&oldid=1048959859, 2021. [Online; accessed 16-October-2021].

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