Learning to play Table Tennis using Multi-agent Reinforcement Learning

Aratrika Basu
University of Southern California
aratrika@usc.edu

Jayantraj Coimbatore Selvakumar
University of Southern California
jcoimbat@usc.edu

Justin Fu
University of Southern California
ziyuefu@usc.edu

Manikanta Chunduru Balaji
University of Southern California
mchundur@usc.edu

Shruthi Ramesh
University of Southern California
shruthir@usc.edu

Sowmya Voona
University of Southern California
voona@usc.edu

Yue Wu
University of Southern California
ywu73061@usc.edu

Abstract—**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(MARL) methods that could well perform in table tennis.**

Index Terms—**Table Tennis, Game Learning, RL, MARL**

I. Introduction

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. 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.

II. Related Works

A. 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 [1]. 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 [2]. With the development of deep learning, neural networks empower RL with unprecedented abilities in field of Go [**?**], Atari [4], StarCraft [5], Robotics [6]. 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 [**?**], which is temporal difference version of policy gradient, A2C [8], which directly performs gradient ascent in asynchronous manner, and PPO [9], which indirectly maximize a surrogate objective function. Another Family is based no action value function, called Q-Learning, first raised by Watkins in 1992 [10].

B. Table Tennis Robots

Attempts to use robots to play table tennis could be traced back to the 80s. Since Anderson [11] 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 [12], [13], [14], [15], [16], [17].

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. [18] demonstrate a model-free approach mixed of evolutionary search and CNN-based policy architectures. Jonas et al. [19] shows a modified DDPG [20] 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 [21].

C. 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 [22], 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 [23]. 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) [22]. 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 [24], [25]. Others try to build communication protocals between agents [26], [27]. Another major MARL learning diagram is *Centralized Training and Decentralized Execution* (CTDE). One represetative CTDE method is MADDPG [28], 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.

III. Data and Environments

The Table Tennis Environment consists of the following components.

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 [**?**]. 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 [**?**].
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

IV. Methods

A. Preliminaries

The Table Tennis Game could be described as a Markov Process, and is a Markov Game [29].

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, $T : S \times A \rightarrow P(S)$ a stochastic transition function, $R : S \times A \rightarrow R$ a reward function, $\lambda \in [0, 1)$ a discount factor. The agent(table tennis player) interacts with the ball by performingits policy $\pi : S \rightarrow P(A)$. 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 $T : S \times A^1 \cdots A^N \rightarrow P(S)$, reward function $R : S \times A^1 \cdots A^N \rightarrow R$. 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 ${r^i}_t = R^i(s_t, a_t^1, \cdots, a_t^N)$.

B. Model

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 consists of table tennis board, two table tennis bats and a ball.We have modeled these game-objects on the unity environment.

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 last the 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.

V. Results and Analysis

A. Modeling and Training

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 [9] 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.
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 [I] contains the hyper parameter combinations.
3) Observation:TODO:Need to Insert ELO diagrams of PPO

2) SAC:

1) Model description: Soft Actor Critic(SAC) is an off-policy model-free reinforcement learning algorithm.

Buffer Size	Learn Rt	Epochs	Learn Sch	Layers	Max Steps	FinaBuEfLerO size	Batch size	Learn Rate	Buffer init steps	Layer	Steps	Final E
2048000	0.0003	3	constant	3	50M	126000000	128	0.0003	0	2	7.5M	2352
20480	0.01	10	linear	3	50M	12050000	128	0.01	0	2	2M	1272
20480	0.01	100	linear	3	50M	126000000	128	0.003	0	2	3.6M	1540
20480	0.01	500	linear	3	50M	1210000000	1024	0.0003	1000	2	4M	2002
20480	0.01	1000	linear	3	50M	126000000	512	0.0003	1000	2	3M	1953
2048000	0.001	3	constant	2	50M	126000000	512	0.0003	1000	2	10M	2130
20480	0.01	3	constant	2	50M	1210000000	1024	0.0003	1000	3	3.9M	1915
20480	0.01	3	constant	3	50M	1210000000	1024	0.0003	1000	3	0.7M	1748
20480	0.03	3	constant	2	50M	126000000	512	0.003	0	3	1.98M	2005
2048000	0.0003	3	constant	2	50M	126000000	512	0.0003	0	2	Prog.	Prog

TABLE I: PPO Hyper parameter Combination

TABLE II: SAC Hyper parameter Combination

This RL algorithm was developed jointly by UC Berkely and Google and was introduced in the year 2018 [9]. It is considered one of the most efficient algorithm to be used in real-world robotics.
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.

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 hidden units in each layer as 256, learning rate schedule as 'constant' and initial ELO as 1200. The table [II] contains the hyper parameter combinations.

3) Observation:TODO:Need to Insert ELO diagrams of SAC once training finishes

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 [**?**]. 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
2) Training :
3) Observation :

VI. Conclusions

References

[1] 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].

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[4] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017, cite arxiv:1710.02298Comment: Under review as a conference paper at AAAI 2018. [Online]. Available: http://arxiv.org/abs/1710.02298

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning." *Nat.*, vol. 575, no. 7782, pp. 350–354, 2019. [Online]. Available: http://dblp.uni-trier.de/db/journals/nature/nature575.html#VinyalsBCMDCCPE19

[6] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation." *CoRR*, vol. abs/1808.00177, 2018. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1808.html#abs-1808-00177

[7] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. [Online]. Available: https://proceedings.mlr.press/v48/mniha16.html

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[10] C. J. C. H. Watkins and P. Dayan, "Technical note q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

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

[12] F. Miyazaki, M. Matsushima, and M. 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.

[13] K. Mülling and J. Peters, *A Computational Model of Human Table Tennis for Robot Application*, 2009, p. 57.

[14] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a mixture of motor primitives," in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, 2010, pp. 411–416.

[15] Y. Huang, D. Buchler, O. Koc, B. Schölkopf, and J. 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*. IEEE, 2016, pp. 650–655. [Online]. Available: https://doi.org/10.1109/HUMANOIDS.2016.7803343

[16] R. Mahjourian, N. Jaitly, N. Lazic, S. Levine, and R. Miikkulainen, "Hierarchical policy design for sample-efficient learning of robot table tennis through self-play." *CoRR*, vol. abs/1811.12927, 2018. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1811.html#abs-1811-12927

[17] K. Muelling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," in *AAAI Fall Symposium on Robots that Learn Interactively from Human Teachers*, 2012, pp. 263–279. [Online]. Available: http://www.aaai.org/ocs/index.php/FSS/FSS12/paper/view/5602

[18] W. Gao, L. Graesser, K. Choromanski, X. Song, N. Lazic, P. Sanketi, V. Sindhwani, and N. Jaitly, "Robotic table tennis with model-free reinforcement learning," 2020.

[19] J. Tebbe, L. Krauch, Y. Gao, and A. Zell, "Sample-efficient reinforcement learning in robotic table tennis," 2021.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[21] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, "A learning approach to robotic table tennis," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 767–771, 2005.

[22] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2021.

[23] Y. Shoham, R. Powers, and T. Grenager, "Multi-agent reinforcement learning: a critical survey," Tech. Rep., 2003.

[24] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning." *CoRR*, vol. abs/1707.04402, 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#PalmerTBS17

[25] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," 2016.

[26] X. Kong, B. Xin, F. Liu, and Y. Wang, "Revisiting the master-slave architecture in multi-agent deep reinforcement learning," 2017.

[27] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning." in *NIPS*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 2137–2145. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2016.html#FoersterAFW16

[28] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments." in *NIPS*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 6379–6390. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2017.html#LoweWTHAM17

[29] M. L. Littman, "Value-function reinforcement learning in markov games." *Cogn. Syst. Res.*, vol. 2, no. 1, pp. 55–66, 2001. [Online]. Available: http://dblp.uni-trier.de/db/journals/cogsr/cogsr2.html#Littman01