1 SOLVING THE TENNIS ENVIRONMENT

1.1 LEARNING ALGORITHM

2 different Multi Agents based learning algorithm have been implemented:

- Multi Agent Deep Deterministic Policy Gradient
- Multi Agent Proximal Policy Optimization

Both are actor-critic based. During training, the input of the critic being enriched with the observations of all agents and their actions. However, during evaluation, the actor critic is just provided with the state as seen by the agent and not all agents.

1.1.1 MULTI AGENT DEEP DETERMINISTIC POLICY GRADIENT

This implementation is using 4 networks

- One local and one target fully connected actor network
- One local and one target fully connected critic network

The pseudo code of the training is as follow and was derived from the MADDPG paper found here:

- Initialize the weights of the local actor and critic network
- Copy these weights into the target actor and critic network
- Initialize a replay buffer
- Repeat the next steps until the environment is solved
    - Initialize a noise generator to disturb the output of the local actor network
    - Receive initial observations state s_1 of each agent
    - Do until one episode is done
        - Generate the actions of the agents from the local actor network and disturb it with the noise generator
        - Execute the actions and observe the rewards and the new states
        - To promote collaboration, ensure that the rewards of the agents are the sum of the rewards received by all agents
        - If the timestep is falling under the update frequency then do the following updates (screenshot from Lilian Weng Blog on Policy Gradient)

Actor update:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{o},a\sim\mathcal{D}}[\nabla_{a_i} Q_i^{\vec{\mu}}(\vec{o}, a_1, \dots, a_N) \nabla_{\theta_i} \mu_{\theta_i}(o_i)|_{a_i=\mu_{\theta_i}(o_i)}]$$

Where $\mathcal{D}$ is the memory buffer for experience replay, containing multiple episode samples $(\vec{o}, a_1, \dots, a_N, r_1, \dots, r_N, \vec{o}')$ — given current observation $\vec{o}$, agents take action $a_1, \dots, a_N$ and get rewards $r_1, \dots, r_N$, leading to the new observation $\vec{o}'$.

Critic update:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\vec{o},a_1,\dots,a_N,r_1,\dots,r_N,\vec{o}'}[(Q_i^{\vec{\mu}}(\vec{o}, a_1, \dots, a_N) - y)^2]$$

$$\text{where } y = r_i + \gamma Q_i^{\vec{\mu}'}(\vec{o}', a_1', \dots, a_N')|_{a_j'=\mu_{\theta_j}'} \quad ; \text{TD target!}$$

where $\vec{\mu}'$ are the target policies with delayed softly-updated parameters.

- Perform a soft update of the target network's parameters from the local network

1.1.2 MULTI AGENT PROXIMAL POLICY OPTIMIZATION

This implementation of Proximal Policy Optimization is using a Critic Network as a baseline. So, the network is composed by:

- A Gaussian Actor Network
- A Fully Connected Critic Network

The particularity of the Gaussian Actor Network is its capacity to create a normal distribution based on a given mean of distribution mu and a standard deviation sigma. That is then sampled to generate the values of the actions. The standard deviation is a trainable parameter of the network set at 1 and the mean of distribution is generated by a Fully Connected Network based on the input states.

During training, the Critic Network's input is also augmented by providing it with the state observed by the agent, the action taken by the agent, the state observed by the other agent, and the action taken by that later.

The pseudo code is as follow:

- Initialize the weights of the network
- Repeat the next steps until the environment is solved
    - Gather N-Trajectories of max_t timesteps where N is the number of parallel agents using the Actor Network
        - Read the state of the environment and request for the actions to be done from the Actor
        - Clip the Actions to [-1,1]
        - Store the states, actions, log prob of the actions, rewards, next state, combined observed states and actions, value of the current state and of the next state as estimated by the Critic Network
    - Calculate the cumulated discounted rewards of each timestep of each agent
        - The return of the last timestep is based on the evaluation of the Critic Network
            - $R_t = r_t + gamma*V(s_{t+1})$
        - The cumulated return of each timestep is
            - $R_t = r_t + gamma*R_{t+1}$
    - Calculate the advantages of each step using the following formulas
        - Taken from the paper [High-Dimensional Continuous Control Using Generalized Advantage Estimation](#) by John Schulman et al.
        - $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$
        - $\sum_{l=0}^{\infty}(\gamma\lambda)^l \tilde{r}(s_{t+l}, a_t, s_{t+l+1}) = \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}^V = \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}$
    - Shuffle the trajectories to break the time correlation and retrieve the trajectories, cumulated rewards, and advantages by batch, then for each batch do the next steps
        - Calculate a clipped surrogate objective using this formula according to the paper [Proximal Policy Optimization Algorithms](#) by John Schulman et al.
        - $r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}$
        - $L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$
        - For each sampled timesteps, calculate the square difference between
            - $R_t = r_t + gamma*R_{t+1}$
            - And $V(s_t)$
        - Calculate the entropy of the probabilities of the sampled timesteps to penalize the actions with high probabilities and avoid the agent to restrict itself on a small set of the environment according to this [answer](#) and this [paper](#)
            - $H(s_t, \pi_\theta(.))$
        - Perform the gradient ascent of
            - $L^{CLIP}(\theta) - c_1(V_\theta(s_t) - R_t)^2 + c_2 H(s_t, \pi_\theta(.))$

1.2 ARCHITECTURE AND HYPERPARAMETERS

Each observation state is composed by 24 elements and each action, 2 elements. Both are continuous values. But the domain of the action is [-1, +1]. The details of both learning algorithm are here below

1.2.1 MADDPG

Random Seed	257	
Optimizer	Adam	
Learning Parameters	0.0003 with 0 weight decay	
Gradient Clipping	5	
Epoch or how many times do we learn from the same collected data	16	
Frequency of the learning phase	Every 32 timesteps	
Frequency of soft transfer	Every 32 timesteps	
Frequency of noise insertion	Every timestep	
Type of noise	Using numpy random	
Scale of the noise	1	
Noise decay	0.9996	
Minimum noise	0.01	
Batch Size	64	
Size of the memory	10,000	
Discounted return factor	0.99	
Size of the bootstrap in TD calculation	4	
Rate of soft update	0.001	
Network Architecture	Actor Network	Critic Network
	Fully Connected (24, 32)	Fully Connected (52, 64)
	ReLU	ReLU
	Fully Connected (32, 32)	Fully Connected (64, 64)
	ReLU	ReLU
	Fully Connected (32, 2)	Fully Connected (64, 1)
	Tanh	

1.2.2 MAPPO

Random Seed	257	
Optimizer	Adam	
Learning Parameters	0.0003	
Gradient Clipping	5	
Epoch or how many times do we learn from the same collected data	3	
Batch Size	256	
Discounted return factor	0.99	
Lambda coefficient in Generalized Advantage Estimation	0.99	

Clipping size	0.1	
C1 or weight of the Critic Network while training the Actor Network in PPO	0.5	
C2 or penalty of actions with higher probability to favor exploration in PPO	0.01	
Network Architecture	Actor Network	Critic Network
	Fully Connected (24, 32)	Fully Connected (52, 32)
	LeakyReLU	ReLU
	Fully Connected (32, 32)	Fully Connected (32, 32)
	LeakyReLU	ReLU
	Fully Connected (32, 2)	Fully Connected (32, 1)
	Tanh	
	Normal Distribution (4, std) where std is a trainable Tensor (4) and the mean is the output of the previous layer	

1.3 RESULTS

- The Multi Agents with PPO solved the environment in 13,012 episodes with an average score of 0.520 in 100 episodes. The agent performed well during evaluation
- The Multi Agents DDPG solved the environment in 3,221 episodes with an average score of 0.508 in 100 episodes. However, during evaluation, it did not reach that score. That could be because of the noise at 0.278 during the last set of training.
- The Multi Agents DDPG was trained more in 1,802 episodes to reach an average of 0.509 in 100 episodes but this time with a minimal noise of 0.01.

The graphs of these results are here below:

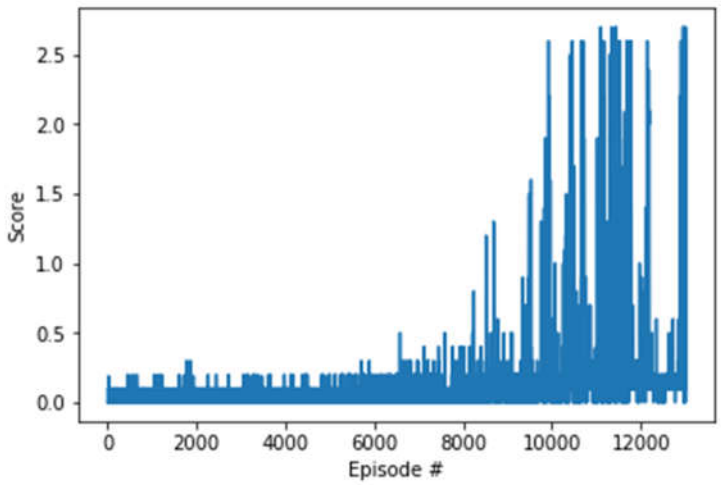

Figure 1 - Scores achieved with Multi Agent PPO

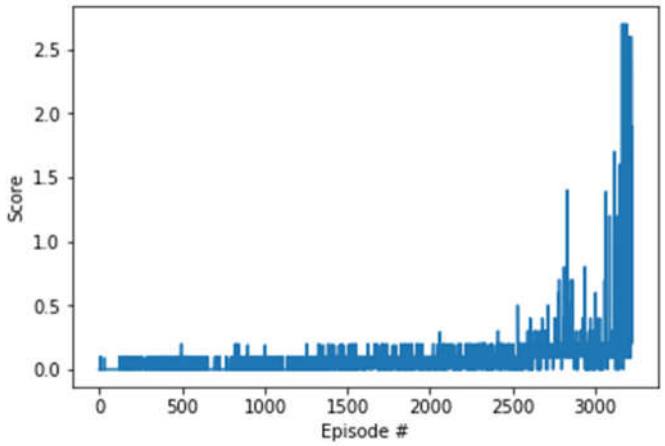

Figure 2 - Scores achieved with Multi Agent DDPG but with a low result in evaluation

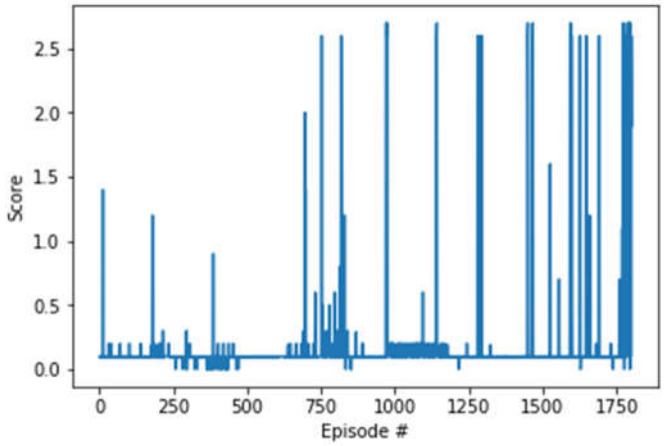

Figure 3 - Scores when continuing the training of the previous MADDPG

1.4 IDEAS FOR FUTURE WORK

- In the implementation of the MADDPG,
  - instead of using a deterministic policy gradient, a Gaussian Network could be used. In that way, the noise generator might not be necessary. Moreover, the entropy of the probabilities of the actions can be utilized to favor the exploration of the actions with low probabilities.
  - A Prioritized Replay could also improve its performance. During visualization of the training, it was noticed that the agent learned to quickly a small portion of the environment. So, in addition to the noise generator, that prioritized replay should force the agent to learn from low occurrence states but with promising result. An implementation can be found here,
- In terms of rewards
  - Only the MADDPG was fed with the sum of the rewards of both agents. So, another improvement of the MAPPO could be achieved in the same way.
  - To train a more aggressive and defensive agent, the rewards can be reviewed so that a -1 on the opponent is calculated as a +1 of current agent. The agent is expected to toss the ball in a way that the opponent cannot get it or at least in a way that the opponent is forced to toss the ball in a way that can be played easily by the current agent.
- The MADDPG was using a ReLU as activation, so it would be interesting to compare the training with a Leaky ReLU
- Finally, applying this learning algorithm to solve planning problem formulated in PDDL should be promising.