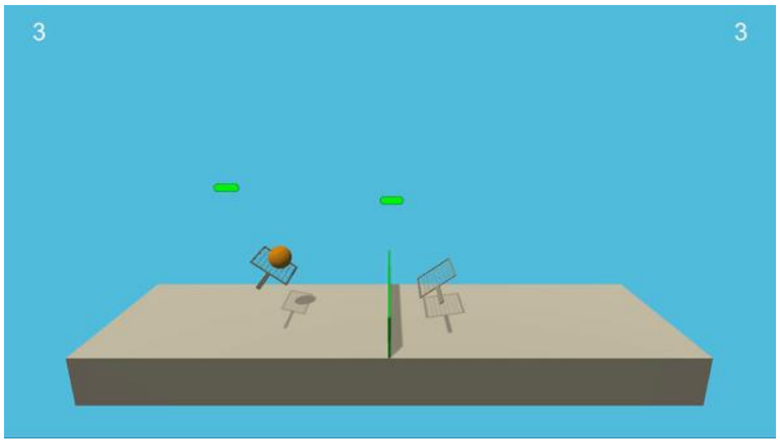
Project 3: Collaboration and Competition

1. Project Introduction

For this project, you will work with the Tennis environment.



In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

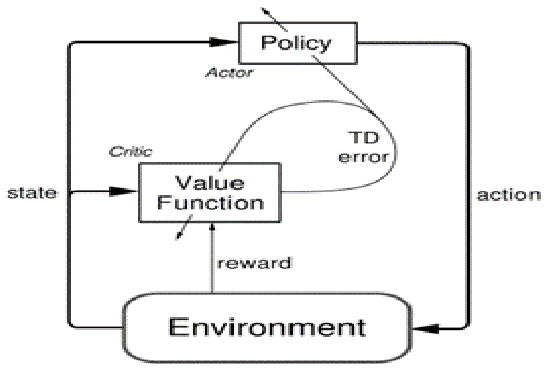
The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

2. Learning Algorithm



To train the agent in this project, we use DDPG algorithm which is introduced as an "Actor-Critic" method (See picture above).

Choosing suitable hyperparameter is important. For this project, following hyperparameter is chosen for the DDPG algorithm.

```
BUFFER_SIZE = int(1e6)      # replay buffer size

BATCH_SIZE = 1024           # minibatch size

GAMMA = 0.99                # discount factor

TAU = 1e-3                  # for soft update of target parameters

LR_ACTOR = 2e-4             # learning rate of the actor

LR_CRITIC = 1e-3            # learning rate of the critic

WEIGHT_DECAY = 0.0001 # L2 weight decay
```

3. Plot of Rewards

This project included 3 Python programs: Tennis.ipynb, ddpg_agent.py, and model.py. After running programs, the program output results are as following:

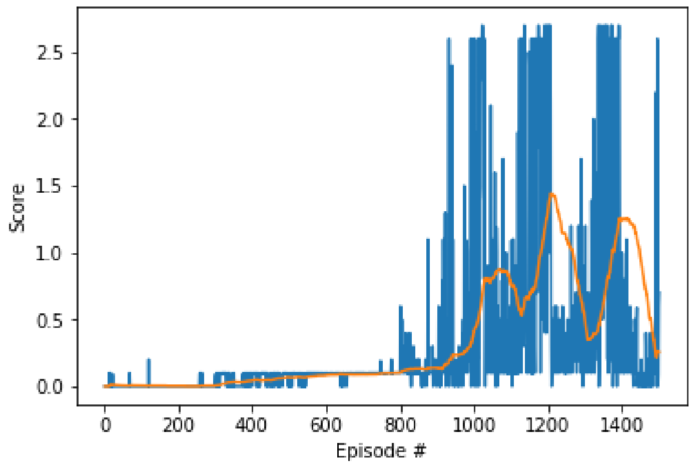
```
Episode:  100   Episode Score: 0.00   Average Score: 0.0048
Episode:  200   Episode Score: 0.00   Average Score: 0.0020
Episode:  300   Episode Score: 0.00   Average Score: 0.0048
Episode:  400   Episode Score: 0.09   Average Score: 0.0439
Episode:  500   Episode Score: 0.00   Average Score: 0.0663
Episode:  600   Episode Score: 0.09   Average Score: 0.0833
Episode:  700   Episode Score: 0.09   Average Score: 0.0880
Episode:  800   Episode Score: 0.10   Average Score: 0.1003
Episode:  900   Episode Score: 0.10   Average Score: 0.1384
Episode: 1000   Episode Score: 0.00   Average Score: 0.4101
Episode: 1011   Episode Score: 2.60   Average Score: 0.5062 <-- Environment s
olved since this episode!
Episode: 1100   Episode Score: 1.09   Average Score: 0.7696
Episode: 1200   Episode Score: 2.60   Average Score: 1.3220
Episode: 1300   Episode Score: 0.10   Average Score: 0.4839
Episode: 1400   Episode Score: 1.00   Average Score: 1.2524
Episode: 1500   Episode Score: 0.70   Average Score: 0.2568

Maximum Average Score (over 100 episodes): 1.4431 at Episode: 1207
```

From the output listing, we can find that the environment solved at Episode 1011. The Average Score is 0.5062. That means: from Episode 912 to Episode 1011 (total 100 Episodes), the average to the 100 scores/rewards are 0.5062. Great than requested value 0.5.

The project’s program continuously calculated for 1500 Episodes. And the Maximum Average Score is obtained as 1.4431 at Episode 1207. After that episode, the situation looks like the over-training because the average score cannot reach to the maximum value then.

Let’s see the output plot now. We have plotted the scores plot from the solution code below. The blue line shows the maximum score over both agents for each episode, and the orange line shows the average score (after taking the maximum over both agents) over the next 100 episodes.



4. Ideas for Future Work

For the ideas to improve the performance of the agent in future, we can use different Actor/Critic network for both the agent instead of common one.

In this project, I tried my best to tune the hyper-parameters, but still not well. I believe that by further tuning hyperparameters precisely, I can improve the performance even more. Same goes for neural network architecture.

Since this is a multi-agent problem, we can try many other permutation and combination of the agents to achieve better results.