MADDPG

This environment is quite interesting compared to single agent environments. It requires the training of two separate agents, and the agents need to collaborate under certain situations (like don't let the ball hit the ground) and compete under other situations (like gather as many points as possible). Just doing a simple extension of single agent RL by independently training the two agents does not work very well because the agents are independently updating their policies as learning progresses. While there are many different RL algorithms for multi-agent settings, for this project I chose to use the Multi Agent Deep Deterministic Policy Gradient (MADDPG) algorithm.

The primary motivation behind MADDPG is that if we know the actions taken by all agents, the environment is stationary even as the policies change, since P(s'|s,a1,a2,π1,π2) = P(s'|s,a1,a2) = P(s'|s,a1,a2,π'1,π'2) for any πi ≠ π'i. This is not the case if we do not explicitly condition on the actions of other agents, as done by most traditional RL algorithms.

This allows the agents to be effectively trained without requiring other agents' observations during inference (because the actor is only dependent on its own observations). Here is the gist of the MADDPG algorithm [1]:

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Multi DDPG agent algorithm with separate replay buffer is used in this project.

Configurations(with hyper parameters):

* 2 hidden layers with 512 and 256 hidden units for both actor and critic

* Shared replay buffer

* Replay batch size 512

* Buffer size 1e6

* Replay without prioritization

* Update frequency 4

* TAU from 1e-3

* Learning rate 1e-4 for actor and 3e-4 for critic

* Ornstein-Uhlenbeck noise

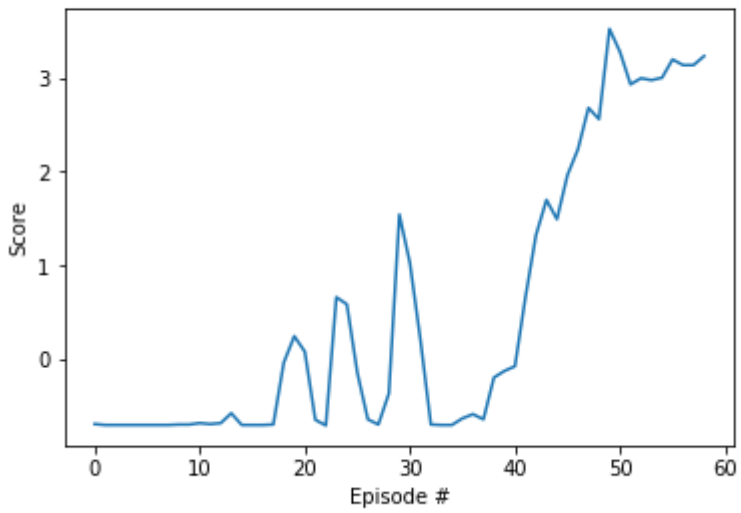
* 20% droput for critic

Plot of Rewards

Plot of rewards can be seen after the environment has been solved.

Environment solved in 59 episodes.

```
Episode 59      Average Score: 0.522
Environment solved in 54 episodes!      Average Score: 0.52
```



Ideas for Future Work

Here's a list of optimizations that can be applied to the project:

1. Build an agent that finds the best hyper parameters for an agent

2. Prioritization for replay buffer

3. Parameter space noise for better exploration

4. Test shared network between agents

5. Test separate replay buffer for agents

References:

1. MADPPG paper - https://arxiv.org/pdf/1706.02275.pdf