1. DESCRIPTION OF THE IMPLEMENTATION:

The problem solved in this project is the Tennis problem as the third project of the Deep Reinforcement learning nanodegree. This problem involves two agents that competes to get a maximum score. The algorithm applied to solve the project is the DDPG (Deep Deterministic Policy Gradient) figure 1.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 1 *DDPG Algorithm*

Each agent applies its own DDPG algorithm independently without considering the other agent. The only form of communication is the reward that each agent perceives.

2. TENNIS SOLUTION

The solution of the Navigation problem is contained in the jupyter notebook named “Tennis.ipynb” This notebook uses two files. The first one contains the code for a multi agent algorithm that uses two ddpg models, this code is contained in the file “MADDPG.py”. The second file called “model.py” contains a code based on neural network to define the actors and critics for DDPG algorithm. The architecture of the neural networks are shown below.

Actor Deep Neural Network				
	Input Size	**Output Size**	**Layer Type**	**Activation Function**
Layer 1	24	256	nn.Linear	Relu
Layer 2	256	256	nn.BatchNorm1d	-
Layer 3	256	256	nn.Linear	Relu
Layer 3	256	2	nn.Linear	Tanh

Table 1 *Actor and Actor target dnn architecture*

Critic Deep Neural Network				
	Input Size	**Output Size**	**Layer Type**	**Activation Function**
Layer 1	24	128	nn.Linear	Relu
Layer 2	128	128	nn.BatchNorm1d	-

Layer 3	130	128	nn.Linear	Relu
Layer 3	128	1	nn.Linear	-

Table 2 *Critic and Critic target dnn architecture*

The hyperparameters are.

Hyperparameters	
Replay buffer size	1e5
Minibatch size	256
GAMMA (discount factor)	0.99
TAU (soft update)	1e-3
Learning rate	2e-4
Weight decay	0

Table 3 *Hyperparameters*

The number of episodes used for training were 1296 episodes and after running the training we observe the following graph that shows how the average reward increases over time. To plot this graph, we only took the max reward of both agents and we calculated the average reward over 100 consecutive episodes. At the end we can see that our algorithm reach the average reward of 0.5 when the problem is considered solved.

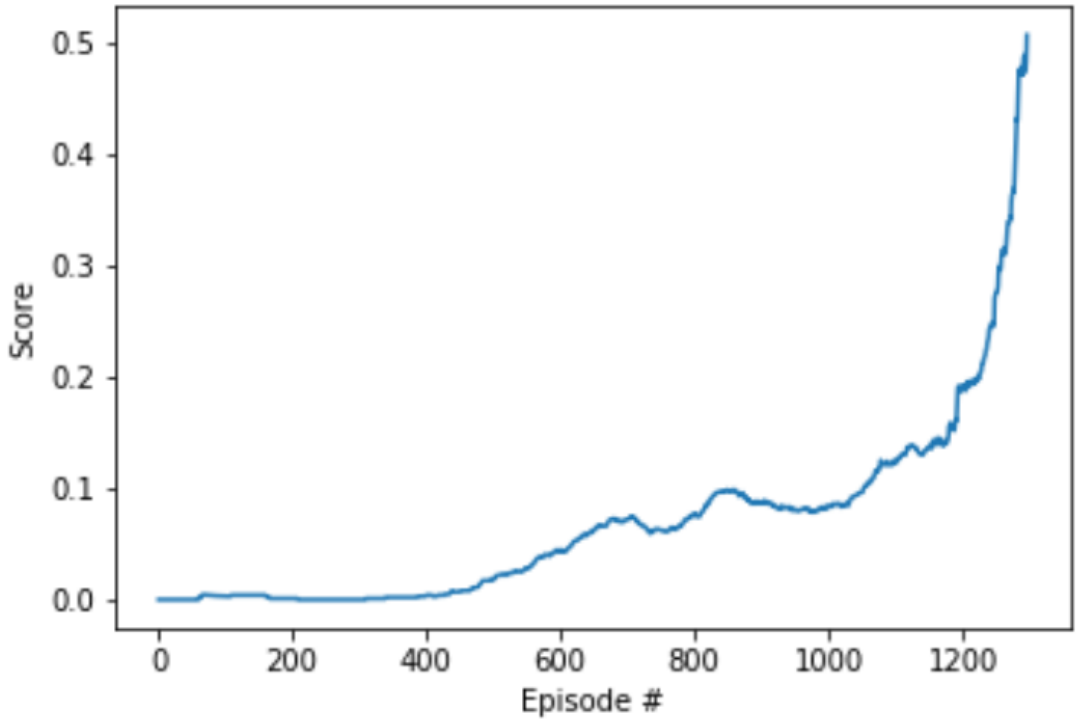

Table 2 *reward plot*

The weights for the trained model are stored in four files, one actor and one critic for each agent. The files are named "checkpoint_actor_agent_0.pth", "checkpoint_actor_agent_1.pth", "checkpoint_critic_agent_0.pth" and "checkpoint_critic_agent_1.pth.pth" that corresponds to the weights of the actor and the critic model.

3. FUTURE WORK

To improve the performance and the training speed of DDPG algorithm GAE (Generalized Advantage Estimation) can be added. There are other algorithms that can be implemented to solve this environment like Proximal Policy Optimization or A2C. Another approach that can be taken for multi agent environments is to coordinate and communicate each agent to have a mixed optimal policy that would consider the joined space of both agents.