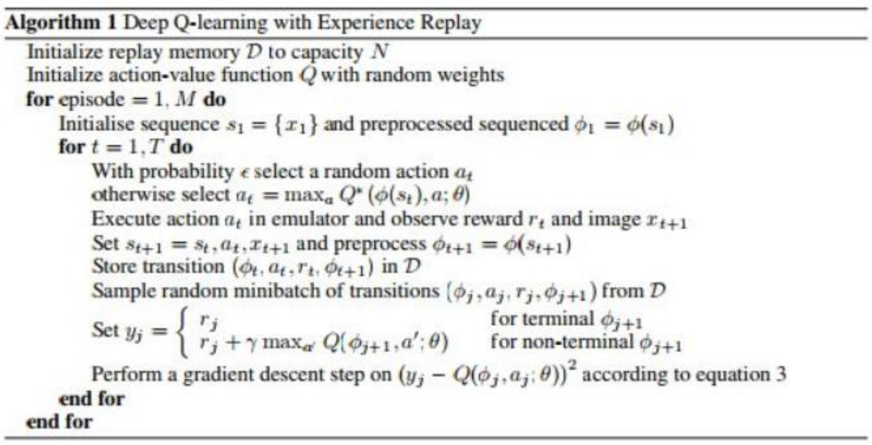
1. DESCRIPTION OF THE IMPLEMENTATION:

The problem solved in this project is the Navigation problem as the first project of the Deep Reinforcement learning nanodegree. The algorithm applied to solve it is the Deep Q Learning algorithm with experience replay as shown in figure 1



Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

Figure 1 *perceptron output*

2. NAVIGATION SOLUTION

The solution of the Navigation problem is contained in the jupyter notebook named "*Navigation.ipynb*" This notebook uses two files the first one contains the code for the deep Q learning agent and the name of this file is "*dqn_agent.py*". The code inside this class also uses another file called *"model.py"* and it uses a deep neural network with three layers and it has the following characteristics.

CNN Architecture		
	Input Size	**Output Size**
Layer 1	37	64
Layer 2	64	64
Layer 3	64	4

Table 1 *Neural network architecture*

The hyper parameters to run the algorithm are.

Hyperparameters	
Epsilon Start	1.0
Epsilon End	0.01
Epsilon Decay	0.995
Learning Rate	5e-4
Discount Factor	0.99
Buffer Size	1e5
Batch Size	64

Table 2 *Hyperparameters values*

The number of episodes used for training were 1000 episodes and after running the training we observe the following graph that shows how the average reward increases over time.

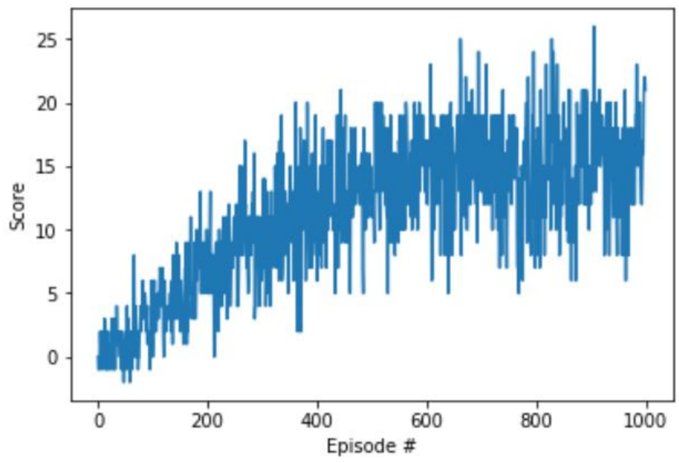


***Figure 2** Average reward for Navigation problem*

The weights for the trained model is stored in the file named *"checkpoint.pth"* you can load the values of the model and run the agent in the environment.

2. NAVIGATION PIXELS SOLUTION

The solution of the Navigation pixels problem is contained in the jupyter notebook named "*Navigation_Pixels.ipynb*". It is worth to mention that the solution for this problem compiles and runs correctly but it was not trained due the lack of computation resources (Feedback for this solution will be appreciated). This notebook also uses two files "*dqn_visual_agent.py*" and *"visual_model.py"* to solve this problem we use a convolutional neural network to read the pixels of the visual state. The architecture of the CNN is

Deep Neural Network Architecture				
	Input Channels	**Output Channels**	**Filter (kernel) Size**	**Stride**
Conv Layer 1	3	32	8	4
Conv Layer 2	32	64	4	2
Conv Layer 2	64	64	3	0
Fully Con Layer 1	3136	512		
Fully Con Layer 2	512	4		

***Table 3** CNN architecture*

The hyper parameters to run the algorithm are the same as in table 2

2. FUTURE WORK

More enhancements can be added to the Deep Q Learning algorithm as the ones mentioned in the Rainbow algorithm and with that we might obtain a better performance of the algorithm in the given environment**.**