# Project 1 Navigation

## Learning Algorithm

Deep Q learning algorithm is used in the implementation. Basicly Model and Agent created in coding exercise is used for the implementation.

Small Changes are applied to the hyper parametes to enhance the performance.

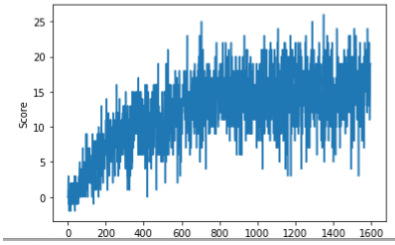
Implementation consistes of following parts

- Model.py: A fully connected deep neural network using Pytorch framework is utlized.
    - State size is a parameter for the construction of the network. For the vectoral Banana implementation state size is 37
    - Architecture of the neural network is as follows
        - An input layer equal to the size of state size 37
        - 2 hidden layers of size 64
        - An output layer equal to the action size
- Dqn_agent.py: A DQN agent with replay buffer is implemented. Following are the parameters used in training
    - BUFFER_SIZE = int(1e5)  # replay buffer size
    - BATCH_SIZE = 64  # minibatch size
    - GAMMA = 0.99  # discount factor
    - TAU = 1e-3  # for soft update of target parameters
    - LR = 5e-4  # learning rate
    - UPDATE_EVERY = 4  # how often to update the network
- Navigation.ipynb : This is the juypter notebook to train and test the agent. Implementation covers following items
    - Necessary imports
    - Initialization of the agent with 37 states and 4 actions
    - Main training loop
    - Testing of the trained agent.

## Results

Following is the results that I have achieved with n_episodes=2000, max_t=2000, eps_start=1.0, eps_end=0.01, eps_decay=0.996 parameters

```
Episode 100     Average Score: 1.52
Episode 200     Average Score: 5.28
Episode 300     Average Score: 8.29
Episode 400     Average Score: 9.46
Episode 500     Average Score: 10.23
Episode 600     Average Score: 11.39
Episode 700     Average Score: 13.61
Episode 800     Average Score: 14.36
Episode 900     Average Score: 14.73
Episode 1000    Average Score: 14.79
Episode 1100    Average Score: 14.77
Episode 1200    Average Score: 14.77
Episode 1300    Average Score: 15.03
Episode 1400    Average Score: 15.13
Episode 1500    Average Score: 14.97
Episode 1598    Average Score: 16.02
Environment solved in 1498 episodes!    Average Score: 16.02
```



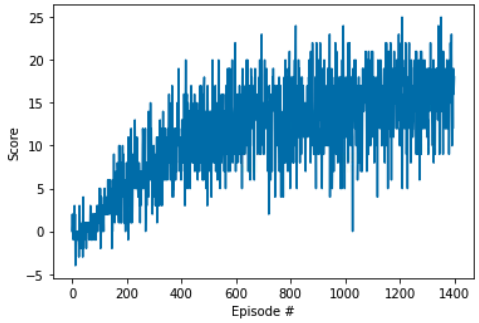

Following are the results I got when I moved to a machine with GPU with following parameters (I have uploaded the version in the GPU based machine to github)

n_episodes=2000, max_t=500, eps_start=1.0, eps_end=0.01, eps_decay=0.996

- BUFFER_SIZE = int(1e6)  # replay buffer size
- BATCH_SIZE = 128        # minibatch size
- GAMMA = 0.99            # discount factor
- TAU = 1e-3              # for soft update of target parameters
- LR = 0.0001             # learning rate
- UPDATE_EVERY = 2        # how often to update the network

```
cuda:0
Episode 100     Average Score: 0.21
Episode 200     Average Score: 3.71
Episode 300     Average Score: 6.77
Episode 400     Average Score: 8.80
Episode 500     Average Score: 11.37
Episode 600     Average Score: 12.26
Episode 700     Average Score: 13.76
Episode 800     Average Score: 13.05
Episode 900     Average Score: 14.15
Episode 1000    Average Score: 13.99
Episode 1100    Average Score: 14.41
Episode 1200    Average Score: 15.68
Episode 1300    Average Score: 15.29
Episode 1400    Average Score: 16.00

Environment solved in 1300 episodes!    Average Score: 16.00
```




You can see how the agent plays the game from the video banana2.mov

When played with the trained agent it was possible for the agent get a score of 16

```
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :

Unity brain name: BananaBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 37
        Number of stacked Vector Observation: 1
        Vector Action space type: discrete
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
```

```
Score: 16.0
```

# Future and Extra Work

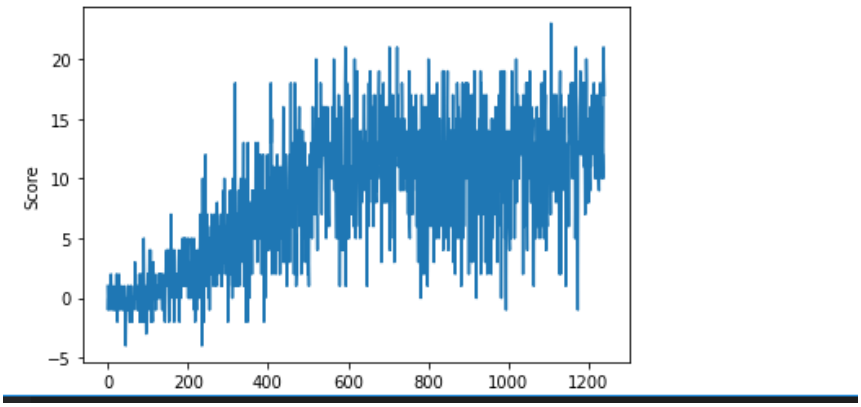
## Extra Work

Just the understand how we can a train a pixel-based agent I have implemented pixel based agent. I have created following files to test the pixel based agent

- Model_pix.py : a convolutional network to get 4 sequential images from the environment.
    - 3 3D convolutions with batch normalization and reLu activation
    - Two fully connected linear layers
- Dqn_agent_pix.py: no major changes other than import of model_pix.
- Navigation_Pixels.ipynb: Major change from above is for collecting four consecutive images to for training and conversion of the frame channel structure to match with the convolutional network.

Following is the output of pixel based training

```
cuda:0
torch.Size([1, 3, 4, 84, 84])
Convolution output size: 2304
---init----
torch.Size([1, 3, 4, 84, 84])
Convolution output size: 2304
---init----
Episode 100     Average Score: -0.01
Episode 200     Average Score: 1.33
Episode 300     Average Score: 3.29
Episode 400     Average Score: 6.16
Episode 500     Average Score: 8.06
Episode 600     Average Score: 10.76
Episode 700     Average Score: 11.82
Episode 800     Average Score: 11.74
Episode 900     Average Score: 11.64
Episode 1000    Average Score: 10.34
Episode 1100    Average Score: 11.34
Episode 1200    Average Score: 12.81
Episode 1240    Average Score: 13.02
Environment solved in 1140 episodes!    Average Score: 13.02
```



After running the trained agent following was the result of the test

```
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :

Unity brain name: BananaBrain
        Number of Visual Observations (per agent): 1
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 0
        Number of stacked Vector Observation: 1
        Vector Action space type: discrete
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
```

```
torch.Size([1, 3, 4, 84, 84])
Convolution output size: 2304
---init----
torch.Size([1, 3, 4, 84, 84])
Convolution output size: 2304
---init----
Score: 8.0
```

## Future work

- Other architectures for CNN can be used
- Double DQN or Duelling DQN can be used