Project 1: Navigation Report

Name: Arun Bala Subramaniyan

Introduction:

In this report, the methodology used for the Navigation project is explained, where the goal is to collect yellow bananas present in the environment. The environment is built with Unity and the complete environment information is provided below.

```
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: , , ,
```

Learning Algorithm:

The DQN algorithm explained in [1] is used to solve the problem to achieve an average score of +13 over 100 episodes. Instead of using convolutional neural network, a simple linear network is used for this problem since there is no information about visual observation. The network is implemented in pytorch and is shown as follows.

```
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

The value for fc1_units and fc2_units are chosen to be 64. The values of training hyperparameters are shown as follows:

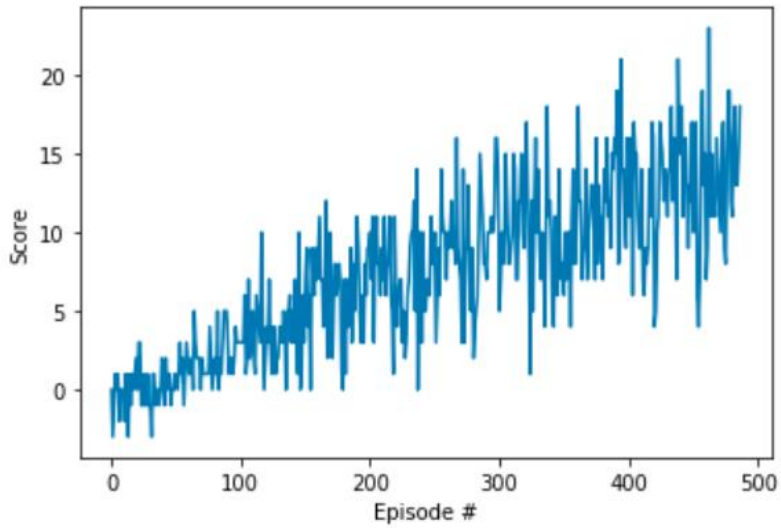
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

Algorithm Explanation:

At first, the algorithm starts by collecting samples from the environment using random actions. Initially, the network weights are randomly assigned and not updated till sufficient samples are available (ie., atleast 64 samples are required to update). There is a experience replay buffer with size of 10000 initialized. All the samples get stored in this buffer and will be used to update the network parameters. A batch size of 64 is passed to the network, randomly obtained from the experience replay buffer. This random sampling allows to eliminate the correlation between samples. Then, the frequency for next update is taken to be 4, ie., the model parameters get updated after continuously collecting 4 samples, storing in the buffer and then randomly selecting 64 samples from the buffer. The neural network has three fully connected layers with RELU activation function. The first layer is fed with the state space of 37 variables and then the last layer gives 4 outputs, corresponding to the action values for each action. The maximum action values are chosen for the corresponding state in this project, ie., actions are chosen greedily. We have two networks defined, i) the local network and ii) target network. The local network parameters are updated every 4 steps and a soft update is made on the target network parameters, ie., {tau*local parameter data + (1.0-tau)*target parameter data}. The value of tau is chosen to be 0.001.

The loss function used in this project is MSE (Mean Squared Error), which is the difference between local and target network values and the objective is to minimize this loss. The optimizer used is Adam optimizer with learning rate of 0.0005. The number of episodes used for training and the maximum time steps for each episode can be varied to observe the learning difference. The problem is considered to be solved if the average reward over 100 episodes reaches a value of +13. Initially 2000 episodes were planned with maximum of 5000 timesteps. But using the hyperparameters and the model, the problem is solved in 387 episodes. The plot of the episode number versus the agent score is given below.

```
Episode 100     Average Score: 0.99
Episode 200     Average Score: 5.00
Episode 300     Average Score: 8.24
Episode 400     Average Score: 10.71
Episode 487     Average Score: 13.02
Environment solved in 387 episodes!     Average Score: 13.02
```



Future Work:

Since the model does not use visual observation, it might not be much suitable for real world implementation. Hence, the banana environment with visual observation needs to be trained and observe the ability of the agent to adapt to the real world. Also, other variants of DQN model can be tried for comparison.

References:

[1] Mnih, V, et al. Human-level control through deep reinforcement learning. *Nature*. 2015, pp: 529-533.