Report for Deep Q-Network Navigation Project

Setting up the environment and DQN agent

Import required libraries

In [1]:
```
from unityagents import UnityEnvironment
import numpy as np
import random
import torch
import numpy as np
from collections import deque
import matplotlib.pyplot as plt
from dqn_agent import Agent
```

Unity environment configuration

In [2]:
```
env = UnityEnvironment(file_name="Banana.app")
```

```
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: , , ,
```

In [3]:

```
# get the default brain
brain_name = env.brain_names[0]
brain = env.brains[brain_name]

# reset the environment
env_info = env.reset(train_mode=True)[brain_name]

# number of agents in the environment
print('Number of agents:', len(env_info.agents))

# number of actions
action_size = brain.vector_action_space_size
print('Number of actions:', action_size)

# examine the state space
state = env_info.vector_observations[0]
print('States look like:', state)
state_size = len(state)
print('States have length:', state_size)
```

```
Number of agents: 1
Number of actions: 4
States look like: [1.         0.         0.         0.         0.8
4408134 0.
 0.         1.         0.         0.0748472  0.         1.
 0.         0.         0.25755    1.         0.         0.
 0.         0.74177343 0.         1.         0.         0.
 0.25854847 0.         0.         1.         0.         0.09355672
 0.         1.         0.         0.         0.31969345 0.
 0.        ]
States have length: 37
```

State and action space and DQN configuration

Action space: This simulation contains a single agent that navigates the environment. It can perform four actions at each time step:

- 0 - walk forward
- 1 - walk backward
- 2 - turn left
- 3 - turn right

State space: The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction

Reward function: A reward of *+1* is provided for collecting a yellow banana, and a reward of *-1* is provided for collecting a blue banana

DQN structure: Similar to Google DeepMind's DQN Nature paper, "Human-level control through deep reinforcement learning" (https://deepmind.com/research/dqn/), the adopted learning algorithm is a vanilla Deep Q-Learning algorithm. As the input vector is a state space instead of raw pixel data, a fully connected layer is used in the first layer instead of a convolutional neural network:

- Fully connected layer 1: with input = 37 state spaces and output = 128 state spaces
- Fully connected layer 2: with input = 128 and output = 64
- Fully connected layer 3: with input = 64 and output = 4, (for each of the 4 actions)

Parameters used in the DQN algorithm:

- Maximum steps per episode: 1000
- Starting epsilion: 1.0
- Ending epsilion: 0.01
- Epsilion decay rates: 0.7, 0.8, 0.9 and 0.99

In [4]:
```
def dqn(agent, n_episodes=5000, max_t=1000, eps_start=1.0, eps_end=
0.01, eps_decay=0.99, train=True):
    """Deep Q-Learning.

    Params
    ======
        n_episodes (int): maximum number of training episodes
        max_t (int): maximum number of timesteps per episode
        eps_start (float): starting value of epsilon, for epsilon-g
reedy action selection
        eps_end (float): minimum value of epsilon
        eps_decay (float): multiplicative factor (per episode) for
decreasing epsilon
    """
    scores = []                        # list containing scores fro
m each episode
    scores_window = deque(maxlen=100)  # last 100 scores
```

```
    eps = eps_start                    # initialize epsilon
    for i_episode in range(1, n_episodes+1):
        env_info = env.reset(train_mode=True)[brain_name]
        state = env_info.vector_observations[0]
        score = 0
        for t in range(max_t):
            action = agent.act(state, eps)
            env_info = env.step(action)[brain_name]
            next_state = env_info.vector_observations[0]   # get next state
            reward = env_info.rewards[0]                   # get reward
            done = env_info.local_done[0]                  # check whether episode has finished
            agent.step(state, action, reward, next_state, done)
            score += reward                                # update the score
            state = next_state                             # roll over the state to next time step
            if done:                                       # exit loop if episode finished
                break
        scores_window.append(score)       # save most recent score
        scores.append(score)              # save most recent score
        eps = max(eps_end, eps_decay*eps) # decrease epsilon
        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores_window)), end="")
        if i_episode % 100 == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores_window)))
        if np.mean(scores_window)>=13.0:
            print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.2f}'.format(i_episode-100, np.mean(scores_window)))
            torch.save(agent.qnetwork_local.state_dict(), 'checkpoint.pth')
            break
    return scores
```

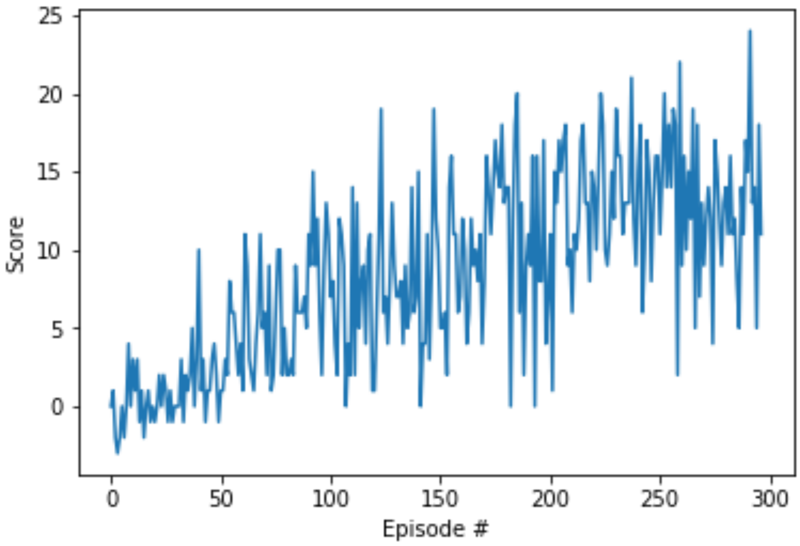
DQN training and results:

Decay rate `eps_decay` = 0.7

In [5]:
```
agent  =  Agent(state_size = state_size, action_size=action_size, s
eed=0)
scores = dqn(agent)

# plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```

```
Episode 100     Average Score: 3.30
Episode 200     Average Score: 8.93
Episode 297     Average Score: 13.01
Environment solved in 197 episodes!     Average Score: 13.01
```

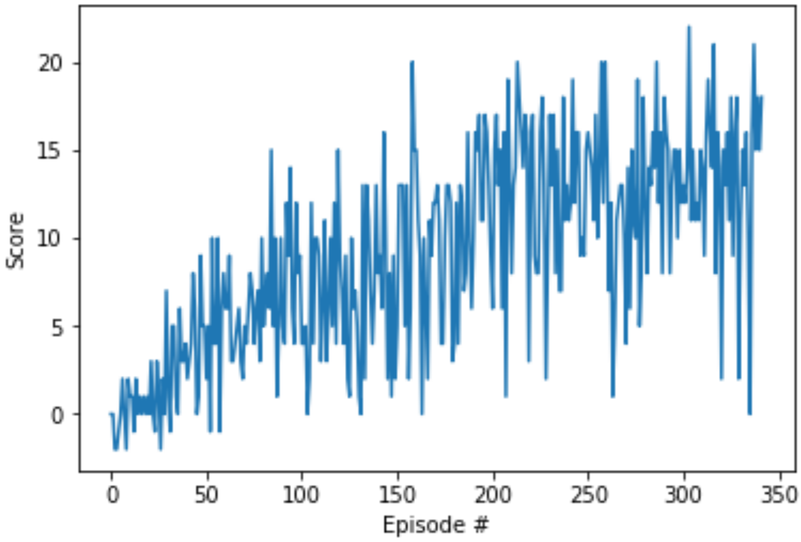


Decay rate `eps_decay` = 0.8

In [5]:
```
agent  =  Agent(state_size = state_size, action_size=action_size, s
eed=0)
scores = dqn(agent)

# plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```

```
Episode 100	Average Score: 3.95
Episode 200	Average Score: 8.49
Episode 300	Average Score: 12.60
Episode 342	Average Score: 13.05
Environment solved in 242 episodes!	Average Score: 13.05
```

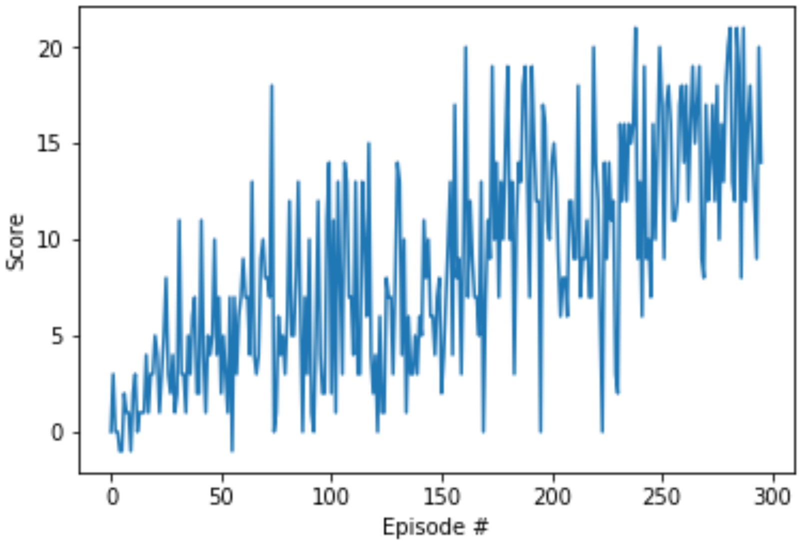


Decay rate eps_decay = 0.9

In [6]:
```
agent  =  Agent(state_size = state_size, action_size=action_size, s
eed=0)
scores = dqn(agent)

# plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```

```
Episode 100	Average Score: 4.51
Episode 200	Average Score: 8.64
Episode 296	Average Score: 13.09
Environment solved in 196 episodes!	Average Score: 13.09
```

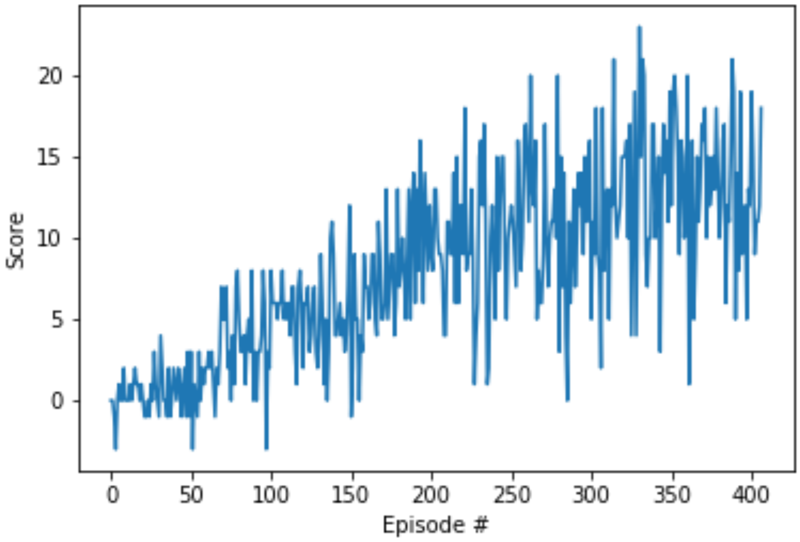


Decay rate `eps_decay` = 0.99

In [5]:
```
agent  =  Agent(state_size = state_size, action_size=action_size, s
eed=0)
scores = dqn(agent)

# plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```

```
Episode 100     Average Score: 1.60
Episode 200     Average Score: 6.61
Episode 300     Average Score: 10.40
Episode 400     Average Score: 12.79
Episode 407     Average Score: 13.12
Environment solved in 307 episodes!     Average Score: 13.12
```



Summary and ideas for future work

By increasing the reward decay rate `eps_decay` closer to *1.0*, the amount of time needed to solve the training criteria increases, while the average score becomes more stable / less fluctuating.

To further enhance the accuracy of the DQN agent, I would recommend implementing additional DQN optimization techniques, such as double DQNs, dueling DQNs or prioritized experience replay

In [6]:
```
env.close()
```

In [ ]: