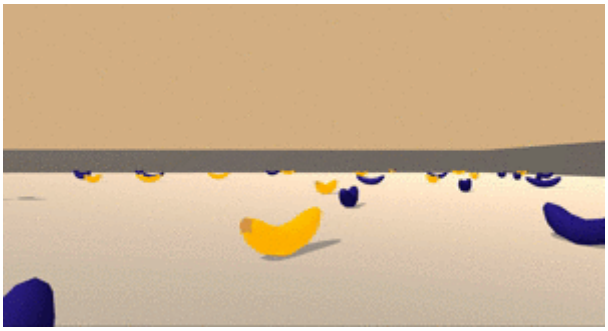
Environment Introduction

For this project, an agent needs to be trained to navigate (and collect bananas!) in a large, square world.



A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

- **0** - move forward.
- **1** - move backward.
- **2** - turn left.
- **3** - turn right.

The task is episodic, and in order to solve the environment, agent must get an average score of +13 over 100 consecutive episodes.

Solving the Environment

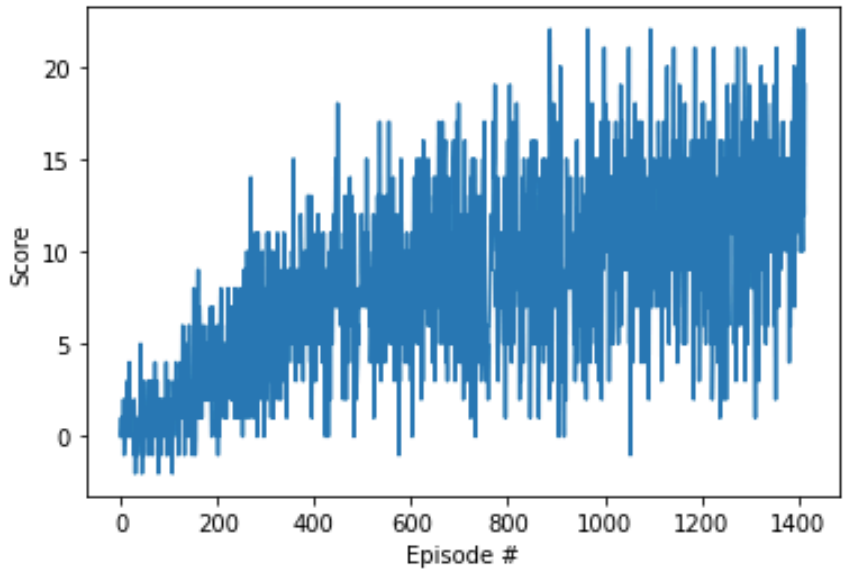
The DQN algorithm with the suggested modification to vanilla DQN agent was used to solve the environment. Initially code was taken from the Deep Q-Networks lessons. First of all, **prioritized replay buffer** implementation was done. This helps in reduction of number of episodes require to train the agent to get the average score to 13. Then to avoid the overestimating the q values **Double DQN network** technique was introduced. And finally, simple Q network architecture was replaced with **Deuling DQN**. With the introduction of Dueling DQN network the number of episodes requires to train the network with a mean score of 13 over consecutive 100 episodes are between 1300 – 1400. No further investigation was done due to limited time to improve the agent further.

- Architecture of Deuling DQN
    - input size = 37
    - output size = 4
    - 2 hidden layers
        - each hidden layer has 128 hidden units and is followed by a ReLU activation layer

    - one value stream network with a single hidden layer (128 hidden units) and is followed by a ReLU activation layer. The output layer of the value stream is followed by a linear activation unit with 1 outputs
    - one advantage stream network with a single hidden layer (128 hidden units) and is followed by a ReLU activation layer. The output layer of the advantage stream is followed by a linear activation unit with 4 outputs
    - addition of state value from value stream to the advantage values from the advantage stream to compute the q values
- Training hyperparameters
    - Replay Buffer size : 50,000
    - Batch size : 128
    - γ : 0.99
    - τ : 0.001
    - learning rate : 0.0005
    - Prioritized Experience Replay, parameter E: 1e-5
    - Prioritized Experience Replay, parameter A: 0.6
    - Prioritized Experience Replay, parameter B: 0.4
    - parameter B increment per learning step : 0.000005

Plot of rewards

```
Episode 100     Average Score: 0.67     Episode Length t : 299
Episode 200     Average Score: 2.63     Episode Length t : 299
Episode 300     Average Score: 4.98     Episode Length t : 299
Episode 400     Average Score: 7.12     Episode Length t : 299
Episode 500     Average Score: 7.88     Episode Length t : 299
Episode 600     Average Score: 8.26     Episode Length t : 299
Episode 700     Average Score: 9.83     Episode Length t : 299
Episode 800     Average Score: 8.93     Episode Length t : 299
Episode 900     Average Score: 10.28    Episode Length t : 299
Episode 1000    Average Score: 9.53     Episode Length t : 299
Episode 1100    Average Score: 11.68    Episode Length t : 299
Episode 1200    Average Score: 11.58    Episode Length t : 299
Episode 1300    Average Score: 11.38    Episode Length t : 299
Episode 1400    Average Score: 12.69    Episode Length t : 299
Episode 1414    Average Score: 13.03    Episode Length t : 299
Environment solved in 1314 episodes!    Average Score: 13.03
```



Ideas for future Work

- ***Learning Faster***: Most of the time was spend on the implementation of the modification of vanilla DQN. Though careful analysis of the code and hyper parameter is required to learn the agent quickly.
- **Other algorithms**: Solving the same environment with other algorithms like, A3C, Distributional DQN, Noisy DQN. Finally this helps to achieve rainbow implementation of DQN