# Learning Algorithm

I used Double Dueling DQN agent for this environment. In order not to overestimate Q values I converted DQN to Double DQN.
Also, agent only get rewards when it collects any bananas. However, when agent running around without collecting bananas, agent still adds these states to experience replay memory. These states are less important than the near moments of collecting bananas. With dueling structure, we can dictate agent that it can give more importance to states of collecting banana moments. Therefore, I converted Double DQN to Dueling Double DQN.

# Hyperparameters

I tried several combinations of hyperparameters as;
Learning Rate: [0.0001, 0.0005, 0.001, 0.005]
Epsilon: [1]
Epsilon Decay: [0.99,  0.995, 0.999]
Minimum Epsilon : [0.01, 0.02]
Batch Size : [64, 128]
Minimum Epsilon : [0.01, 0.02]
Gamma : [0.99, 0.95]
TAU : [0.001]
Buffer Size = 100000
Update every 4 states

Most successful parameters are;

Learning Rate: 0.0005
Epsilon: 1
Epsilon Decay: 0.99
Minimum Epsilon : 0.01
Batch Size : 64
Minimum Epsilon : 0.01
Gamma : 0.99
TAU : 0.001
Buffer Size = 100000
Update every 4 states

# Model Structure

There is a pre-trained model which is Double Dueling DQN with:

- Base Network with two paralel streams
  - Value stream
  - Advantage stream

Base Network has;
- Fully Connected Layer (state_size, 64)
- ReLU
- Fully Connected Layer (64, 64)
- ReLU
- Fully Connected Layer (64, 64)
- ReLU

Value Network has;
- Fully Connected Layer (64, 64)
- ReLU

- Fully Connected Layer (64, 1)

Advantage Network has;
- Fully Connected Layer (64, 64)
- ReLU
- Fully Connected Layer (64, action_size)

Even though I have tried different hidden layer sizes, above was the most successful for tried hyperparameter combinations

I used Adam which was more effective while learning process that results faster training.

I used MSE loss function.

# Forward Action

When we want to return expected Q-values we should calculate them as;

Q(s) = V(s) + (Adv(s) – mean(Adv(s)))

This calculation comes from Dueling addition to DQN.

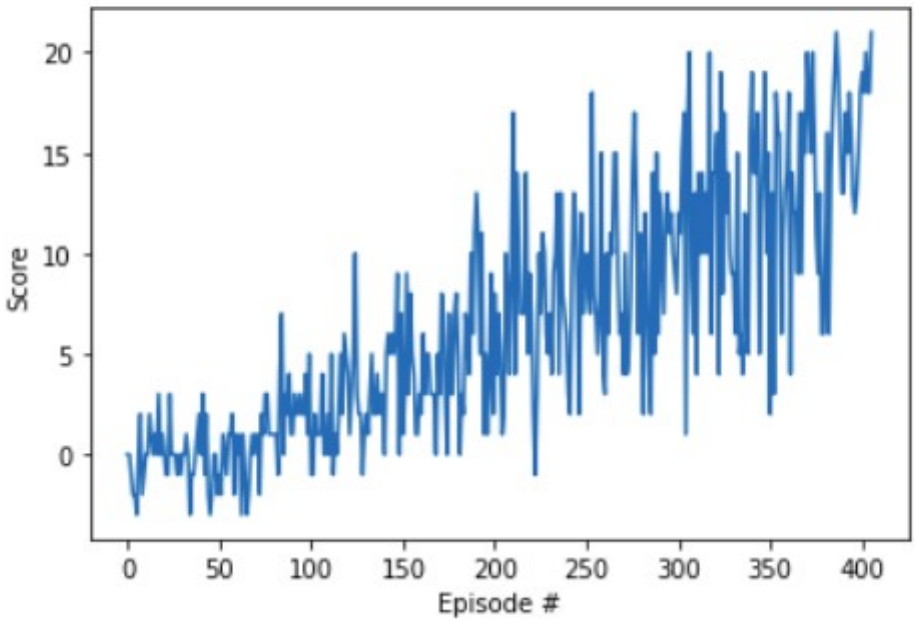
I have tried to add Softmax to the last layer but in my opinion softmax decrease the importance of making mistake or gaining reward effect by clipping between [0,1]. I saw that learning is much slower than model without softmax with the same hyperparameters.

# Plots

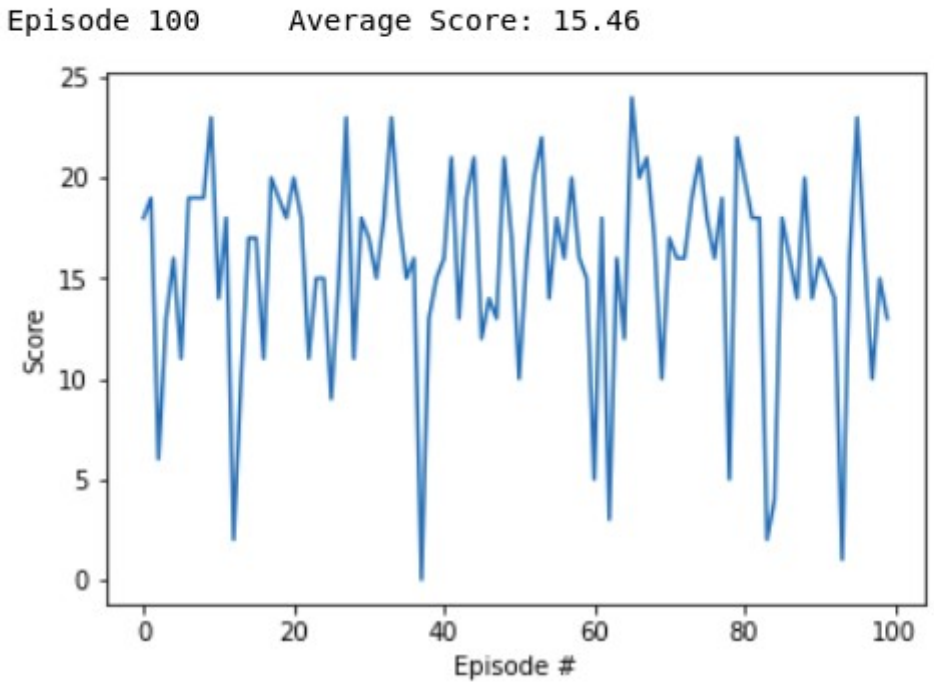
## Training process result:

```
Episode 100     Average Score: 0.45
Episode 200     Average Score: 3.78
Episode 300     Average Score: 8.27
Episode 400     Average Score: 12.59

Environment solved in 406 episodes!     Average Score: 13.00
```



### Testing process result:




# Future Work

We can see that hitting or get so close to a wall is not a very common state. Therefore, at these states agent is not very trained. Maybe if I use priotized experience replay, I can give more importance to these states and that results a better training overall.
Converting learning algorithm to Rainbow, but this kind of not so complex environment, it may be not necessary. However, even unit training speed slows down, I can get better results.