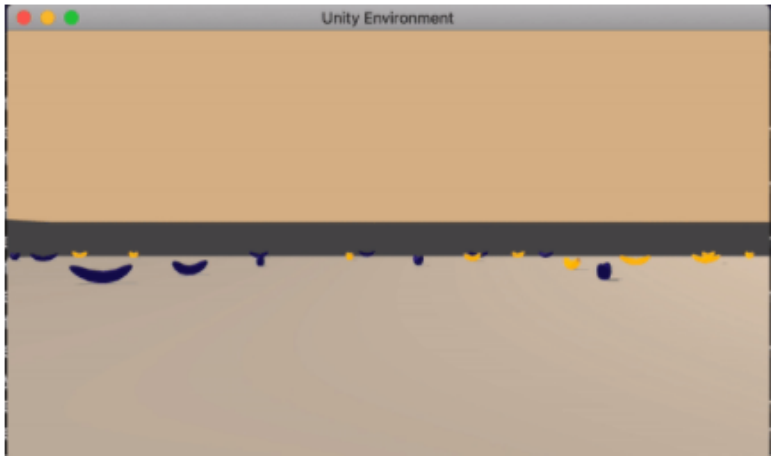
Project 1: Navigation

In this report, I am going to present about the environment and the algorithms that used to solve the navigation problem where the agent is to collect bananas. The project is to solved continuous state space of 37 dimensions, with the goal to collect yellow banana (maximizing the reward) and avoiding the blue banana (minimizing the reward). And there are 4 actions that can be choosed. Picture below is environment representation:



Learning Algorithm

This project use Deep Q-Learning as the learning algorithm. This method is based on Q-learning (temporal difference learning)

The Big idea of temporal difference learning is learn from every experience!

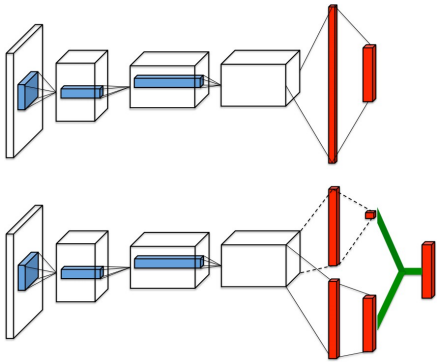
- Update V(s) each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

Not like Monte-Carlo methods, the Q-learning from TD learning can learn from each step, we use current Q-Value of the states to estimate future rewards.

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

We try to use Double DQN that already improve from original DQN algorithm. The idea of Double DQN is to disentangle the calculation of the Q-targets into finding the best action and then calculating the Q-value for that action in the given state. The trick then is to use one network to choose the best action and the other to evaluate that action. The intuition here is that if one network chose an action as the best one by mistake, chances are that the other network wouldn't have a large Q-value for the sub- optimal action. The network used for choosing the action is the online network whose parameters we want to learn and the network to evaluate that action is the target network described earlier.



Result

For training process the DQN need 361 episodes to finish

```
In [10]: # train the agent
         scores = dqn(n_episodes, max_t, eps_start, eps_end, eps_decay)
```

```
Episode 100     Average score:  0.89
Episode 200     Average score:  4.92
Episode 300     Average score:  8.88
Episode 400     Average score:  11.86
Episode 461     Average score:  13.04
Environment solved in  361 episodes!    Average Score:  13.04
```

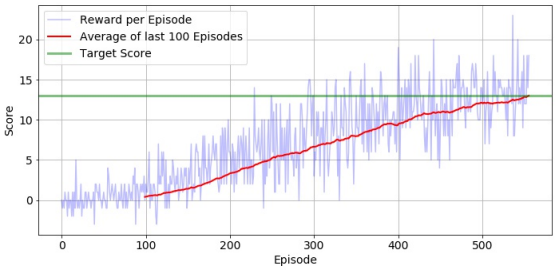
Instead for Double DQN need 456 episodes

```
In [20]: # train the agent
         scores = ddqn(n_episodes, max_t, eps_start, eps_end, eps_decay)
```

```
Episode 100     Average score:  0.41
Episode 200     Average score:  3.29
Episode 300     Average score:  6.67
Episode 400     Average score:  9.30
Episode 500     Average score:  12.14
Episode 556     Average score:  13.01
Environment solved in  456 episodes!    Average Score:  13.01
```

The best performance was achieved by double dqn with score 16 compare to original dqn with score 6

DQN



Double DQN

Performance

Below is performance for DQN and Double DQN

```
In [15]: env_info = env.reset(train_mode=False)[brain_name] # reset the environment
         state = env_info.vector_observations[0]            # get the current state
         score = 0                                          # initialize the score
         while True:
             action = agent.act(state)                      # select an action
             env_info = env.step(action)[brain_name]        # send the action to the environment
             next_state = env_info.vector_observations[0]   # get the next state
             reward = env_info.rewards[0]                   # get the reward
             done = env_info.local_done[0]                  # see if episode has finished
             score += reward                                # update the score
             state = next_state                             # roll over the state to next time step
             if done:                                       # exit loop if episode finished
                 break

         print("Score: {}".format(score))
```

```
Score: 6.0
```

In [24]:
```
env_info = env.reset(train_mode=False)[brain_name] # reset the environment
state = env_info.vector_observations[0]            # get the current state
score = 0                                          # initialize the score
while True:
    action = agent.act(state)                      # select an action
    env_info = env.step(action)[brain_name]        # send the action to the environment
    next_state = env_info.vector_observations[0]   # get the next state
    reward = env_info.rewards[0]                   # get the reward
    done = env_info.local_done[0]                  # see if episode has finished
    score += reward                                # update the score
    state = next_state                             # roll over the state to next time step
    if done:                                       # exit loop if episode finished
        break

print("Score: {}".format(score))
```

```
Score: 14.0
```

Ideas for future works

- To improve the performance, I am planning to Dueling DQN, Prioritized Experienced Replay.
- Train network from image data directly