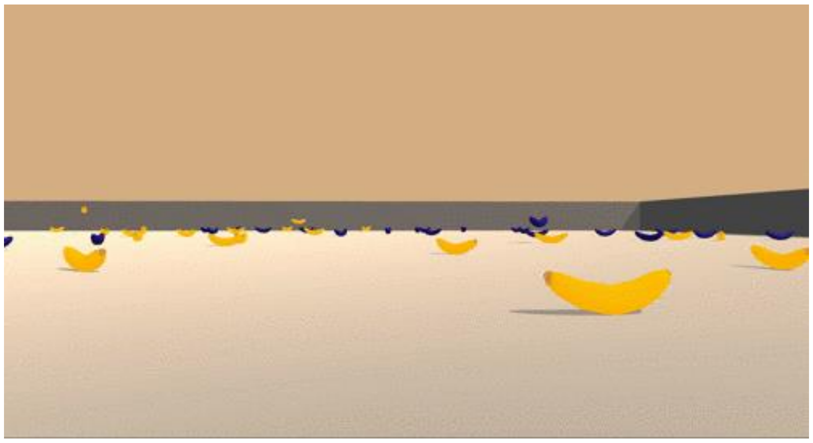
Project 1: Navigation Overview



Overview: Train an agent to navigate a large world and collect yellow bananas, while avoiding blue bananas.

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 the 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, the agent must get an average score of +13 over 100 consecutive episodes.

Deep Q Network

The goal of the agent is to select actions in a fashion that maximizes cumulative future reward. This uses a deep convolutional neural network to approximate the optimal action-value function.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\, a_t = a,\, \pi\right]$$

which is the maximum sum of rewards r_t discounted by $^{\gamma}$ at each timestep ^{t}, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a).[1] The DQN acts as a function approximator, where the agent tries to determine the optimal action value function Q.

DQN Learning Algorithm

Using pytorch, the model I used consisted of two 64 hidden linear layers with ReLu. The input layer consisted of 37 neurons and the output layer consisted 4 nodes based on the action size.

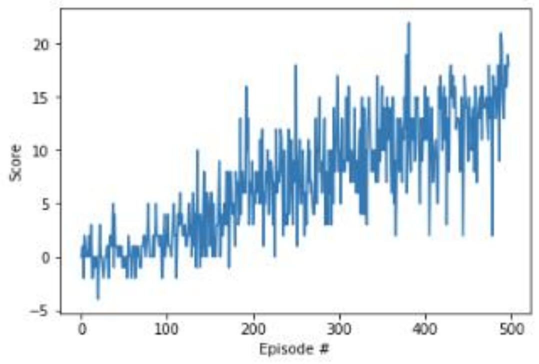
The hyperparameters that were used in the model consisted of:

```
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
n_episodes=2000             # Max Number of Episodes
max_t=1000                  # Max Number of timesteps per episode
eps_start=1.0               # Starting value of Epsilon, for epsilon-greedy action selection
eps_end=0.01                # Min value of Epsilon
eps_decay=0.995             # Epsilon Decay
```

Plot of Rewards

The DQN achieved an average score of 13 in 389 episodes as seen below:

```
Episode 100     Average Score: 0.51
Episode 200     Average Score: 4.03
Episode 300     Average Score: 7.34
Episode 400     Average Score: 10.56
Episode 498     Average Score: 13.00
Environment solved in 398 episodes!     Average Score: 13.00
```



DQN Improvements

Deep Q-Learning tends to overestimate action values. Double Q-Learning has been shown to work well in practice to help with this. Currently, in order to determine which states are (or are not) valuable, we have to estimate the corresponding action values for each action. However, by replacing the traditional Deep Q-Network (DQN) architecture with a dueling architecture, we can assess the value of each state, without having to learn the effect of each action. [2]

References

[1] https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
[2] Udacity Deep Reinforcement Learning Nanodegree Lesson 2 – Deep Q-Networks Deep Q-Learning