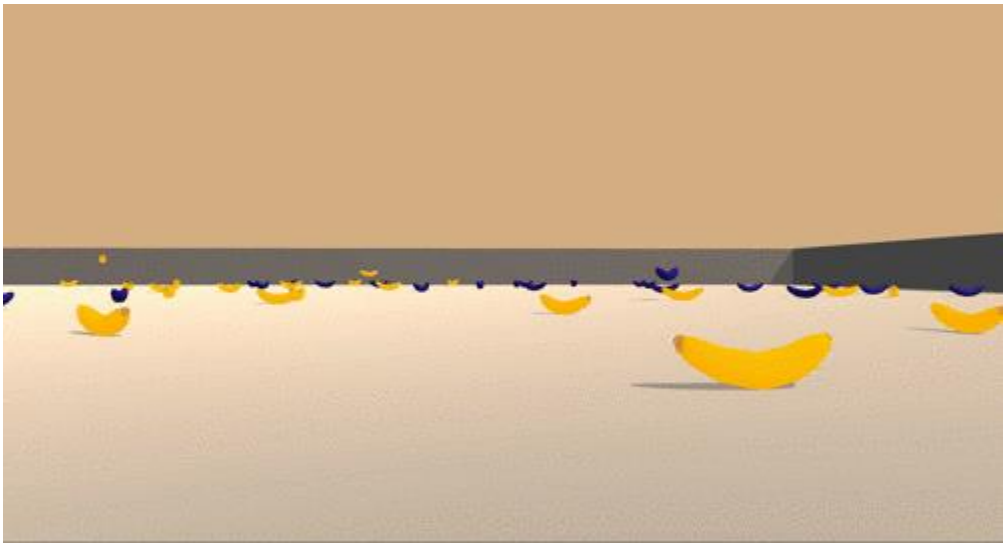
The Environment:

For this project, you will train an agent 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 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, your agent must get an average score of +13 over 100 consecutive episodes.

Architecture:

- DuelingQNetwork consists of two fully connected (FC) hidden layers with 64 nodes connected to the input layer and 64 nodes connected to output layer. Both of the FC layers have RELU activation function. The input has 37 nodes and the output has 4 nodes. The two hidden layers and the RELU activation function are chosen to achieve non-linearity.
- The class DuelingQNetwork is implemented in model.py
- The class Agent and ReplayBuffer is implemented in dqn_agent.py

- Navigation.ipyb has instance of Agent and UnityEnvironment. The Navigation is responsible to train and valid the agent.
- The class Agent has instance of ReplayBuffer and DuelingQNetwork.

Following techniques were used to enhance the learning capability/stability:

- **Experience Replay**: The class ReplayBuffer is responsible for adding and sampling of experience.
- **Fixed Q-Target**: The function approximators are defined in class Agent. They are initialized in constructor as, network and network_target. The network_target approximator is used for finding TD target and the network approximator is used for finding TD expected value. Before starting new time step, the network_target is updated in accordance with network approximator.

```
self.network = DuelingQNetwork(state_size, action_size, seed).to(device)
self.network_target = DuelingQNetwork(state_size, action_size, seed).to(device)
```

- **Double DQN**: Since we are using Fixed Q-Target, the network approximator where used to find best action and for evaluation network_target approximator was used.

```
best_actions = self.network(states).max(1)[1].unsqueeze(1)
Q_targets_next = self.network_target(next_states).gather(1, best_actions)
```

- **DuelingQNetwork**: The class is defined in model.py

Hyperparameters:

Parameter	Value	Description
BUFFER_SIZE	int(1e5)	replay buffer size
BATCH_SIZE	64	minibatch size
GAMMA	0.99	discount factor
TAU	1.00E-03	for soft update of target parameters
LR	5.00E-04	learning rate
UPDATE_EVERY	4	how often to update the network
n_episodes	5000	number of episode
max_t	2000	maximum time stamp
eps_start	1	starting epsilon (for epsilon-greedy policy)
eps_end	0.1	ending epsilon
eps_decay	0.995	rate of decresase epsilon

Future ideas for improving the agent's performance:

- **Prioritized Experience Replay** could be implemented to further enhance the performance of the agent. Its suggested to use '**Sum Tree**' data structure as it helps in a faster implementation of Prioritized Experience Replay.