1. Environment

We need to train an agent to navigate and collect yellow bananas. For every yellow banana collected the reward is +1 and for every blue banana the reward is -1. The state space is 37 dimensions. It includes the agent's velocity and ray-based perception of objects in the front of the agent. The agent has a total of 4 actions. The agent can move forward, backward, left, right, and these actions are indicated in numerical values 0, 1, 2, 3. The problem is considered solved if the average reward is at least 13+ over 100 episodes.

2. Algorithm

We use Deep Q-Network (DQN) to implement reinforcement learning. DQN uses a deep neural network instead of a Q-table. The main features of DQN is that it has an experience replay buffer, local and target network, and uses deep neural networks to create these networks. The experience replay buffer selects training data randomly and removes correlation. The target network and local network is updated periodically so that learning is more stable.

For the training episodes, the agent runs maximum of 2000 episodes. In each episodes, the agent selects the best action or a random action depending on the probability. Once the action is taken, the environment returns next state, reward, done. The agent updates its memory by adding the state, next state, reward, and done to the experience replay buffer. After a specific number of steps, the agent learns from the experiences and updates its local and target network. It randomly selects a number of data in the experience replay buffer to train the agent. Based on the target value and the expected action value, the agent updates the local network. Based on the local network, the agent soft updates the target network. The agent keeps taking steps until the environment notifies the agent that it is done. Training ends when the agent's average score is more than 15.0

For the test episodes, the agent runs maximum of 200 episodes and takes the average of the scores. Unlike training, the agent doesn't take random actions and doesn't update the neural networks. For each step, the agent takes the action with the largest action value.

3. Hyper parameters

The following are the hyper parameters
In dqn_agent.py,

```
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
```

In main.py,

```
Train = 1                    # training mode or testing mode
n_episodes                   # maximum number of episodes
max_t                        # maximum number of steps
```

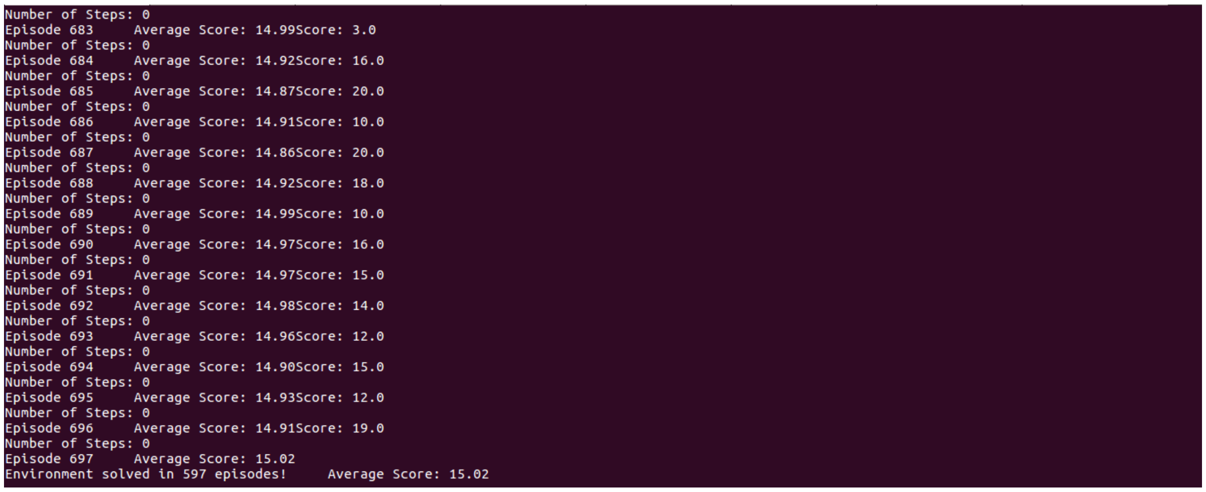
eps   # used to decide explore or take the best action

4. Architecture

The deep neural network model used in the DQN agent is described in model.py. It takes the states as an input of the deep neural network model. Since the state space is 37, the input layer will have 37 nodes. The input layer is connected to two layers with 64 hidden nodes sequentially. The last 64 hidden layers connected to the output layer. The output layer outputs the action value for each action. Since there are 4 actions, the output layer will contain 4 nodes.

5. Results & Figures

When training the agent, the target average score was set to 15.0. This was because when testing, the score had to be at least 13.0 or higher for over 100 steps. Setting the target average score to 15.0 seemed like a safe choice. The agent achieved an average score higher than 15.0 after 697 episodes. The figure below shows the average scores for training and that the training stopped at episode 697.



```
Number of Steps: 0
Episode 683     Average Score: 14.99Score: 3.0
Number of Steps: 0
Episode 684     Average Score: 14.92Score: 16.0
Number of Steps: 0
Episode 685     Average Score: 14.87Score: 20.0
Number of Steps: 0
Episode 686     Average Score: 14.91Score: 10.0
Number of Steps: 0
Episode 687     Average Score: 14.86Score: 20.0
Number of Steps: 0
Episode 688     Average Score: 14.92Score: 18.0
Number of Steps: 0
Episode 689     Average Score: 14.99Score: 10.0
Number of Steps: 0
Episode 690     Average Score: 14.97Score: 16.0
Number of Steps: 0
Episode 691     Average Score: 14.97Score: 15.0
Number of Steps: 0
Episode 692     Average Score: 14.98Score: 14.0
Number of Steps: 0
Episode 693     Average Score: 14.96Score: 12.0
Number of Steps: 0
Episode 694     Average Score: 14.90Score: 15.0
Number of Steps: 0
Episode 695     Average Score: 14.93Score: 12.0
Number of Steps: 0
Episode 696     Average Score: 14.91Score: 19.0
Number of Steps: 0
Episode 697     Average Score: 15.02
Environment solved in 597 episodes!     Average Score: 15.02
```

Fig 1. Average Scores when Training the Agent

The figure below plots the score for each episode during training. We can see that scores are generally higher for later episodes. This means that the agent is learning from the experiences.

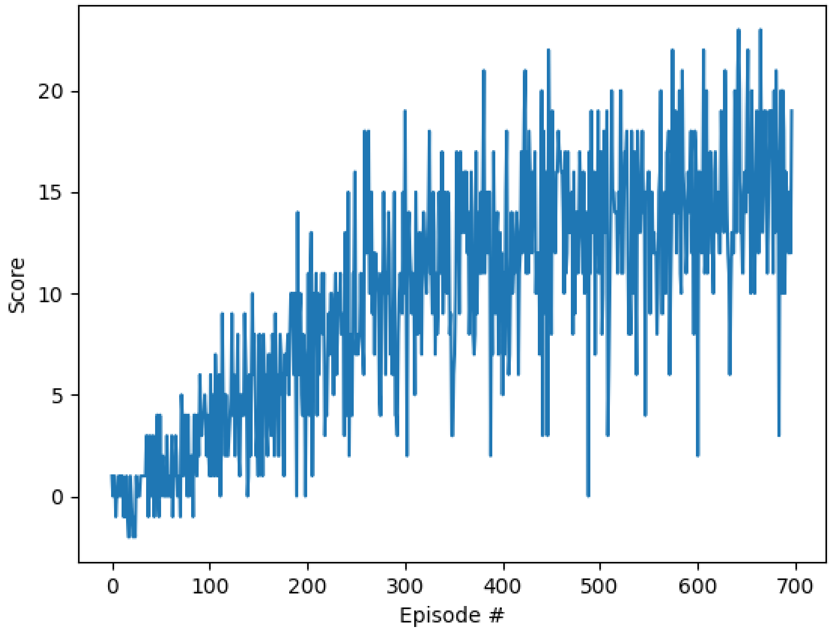
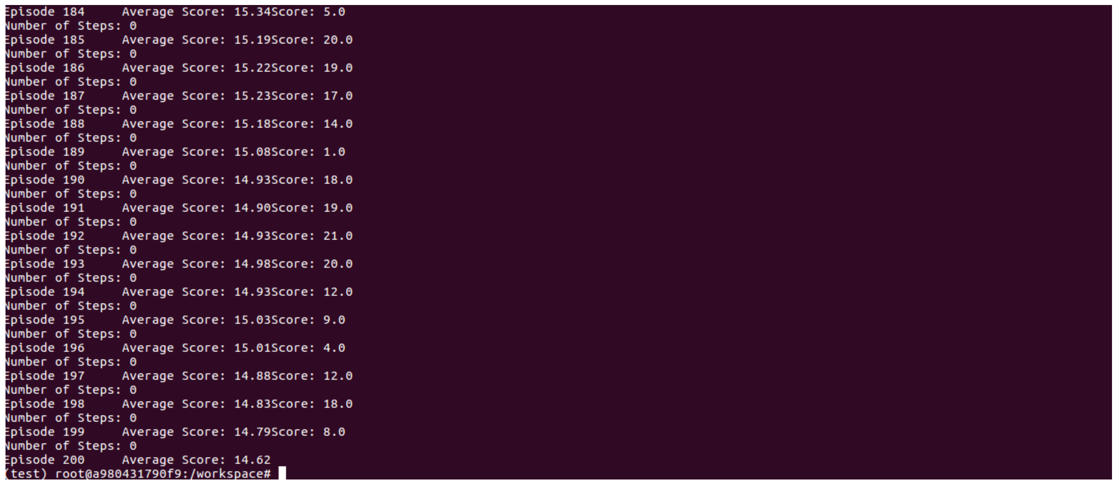


Fig 2. Scores while Training the Agent

To test the agent, we ran the trained agent over 200 episodes and took the average. In the figure below, we can see that the average score was 14.62 after 200 episodes. 14.62 is greater than 13.0. This means that the agent is trained properly and solves the problem.



```
Episode 184     Average Score: 15.34Score: 5.0
Number of Steps: 0
Episode 185     Average Score: 15.19Score: 20.0
Number of Steps: 0
Episode 186     Average Score: 15.22Score: 19.0
Number of Steps: 0
Episode 187     Average Score: 15.23Score: 17.0
Number of Steps: 0
Episode 188     Average Score: 15.18Score: 14.0
Number of Steps: 0
Episode 189     Average Score: 15.08Score: 1.0
Number of Steps: 0
Episode 190     Average Score: 14.93Score: 18.0
Number of Steps: 0
Episode 191     Average Score: 14.90Score: 19.0
Number of Steps: 0
Episode 192     Average Score: 14.93Score: 21.0
Number of Steps: 0
Episode 193     Average Score: 14.98Score: 20.0
Number of Steps: 0
Episode 194     Average Score: 14.93Score: 12.0
Number of Steps: 0
Episode 195     Average Score: 15.03Score: 9.0
Number of Steps: 0
Episode 196     Average Score: 15.01Score: 4.0
Number of Steps: 0
Episode 197     Average Score: 14.88Score: 12.0
Number of Steps: 0
Episode 198     Average Score: 14.83Score: 18.0
Number of Steps: 0
Episode 199     Average Score: 14.79Score: 8.0
Number of Steps: 0
Episode 200     Average Score: 14.62
(test) root@a980431790f9:/workspace#
```

Fig 3. Average Scores for Trained Agent

The figure below plots the score for each episode during testing. There are outliers where the agent scores lower than 10.0. However, we can see that scores are generally higher than 13.0. Compared to Fig 2, Fig 4 doesn't start with low scores and gradually increases. This is because the agent used in Fig 4 is already trained and is not updated.

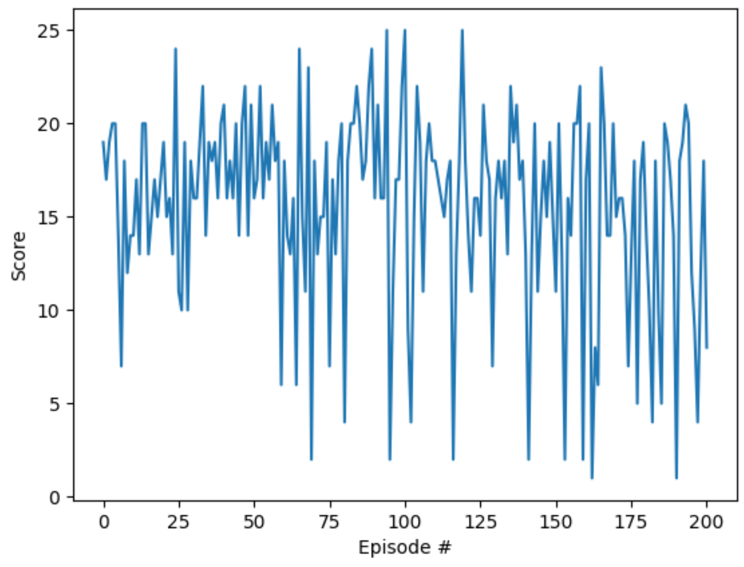


Fig 4. Scores with the Trained Agent

6. Future Idea

Improve installation - We used both docker and a virtual environment. Both docker and virtual environment isolate the environment so it is redundant. In this project, docker was used to install the nvidia drivers, CUDA, Cudnn and pytorch. Installing python 3.6 on the docker was a bit difficult and ended up using a virtual environment to install python 3.6. In future work, only docker should be used and the docker should have nvidia drivers, CUDA, Cudnn, pytorch and python 3.6.

Configure the hyper parameters - In future work, more testing should be done with the hyper parameters. Some of the hyper parameters directly influence the agent’s learning behavior. Each hyper parameter should be tested in the future.

Training with a higher number of episodes - In future work, we should check if performance increases as the number of training episodes increase. We ended the training after it reached 15.0. In future work, we should set a higher target score and train the agent.

Attempt the optional challenge - In the original challenge, state space is 37 dimensions. It includes the agent’s velocity and ray-based perception of objects in the front of the agent. In the optional challenge, the state is a 84x84 RGB image. In the future, to solve this problem, the network model of the DQN agent needs to be configured. We need to test out which model configuration would work better. We can try transforming the 84x84 RGB image to a linear input. We can also try using a convolutional network.