Project: Navigation (Part of Udacity Deep Reinforcement Learning Nanodegree)
by Michael Strobl

Used Neural Network:

```
def __init__(self, state_size, action_size, seed, fc1_units=64, fc2_units=64):
    """Initialize parameters and build model.
    Params
    ======
        state_size (int): Dimension of each state
        action_size (int): Dimension of each action
        seed (int): Random seed
        fc1_units (int): Number of nodes in first hidden layer
        fc2_units (int): Number of nodes in second hidden layer
    """
    super(QNetwork, self).__init__()
    self.seed = torch.manual_seed(seed)
    self.fc1 = nn.Linear(state_size, fc1_units)
    self.fc2 = nn.Linear(fc1_units, fc2_units)
    self.fc3 = nn.Linear(fc2_units, action_size)
```

- Fully connected layer - input: 37 (state size), output: 64
- Fully connected layer - input: 64 (fc1_units), output 64 (fc2_units)
- Fully connected layer - input: 64 (fc1_units), output: 4 (action size)

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 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.

Used Learning Algorithm:

Deep Q Learning (DQN) with Replay Buffer

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

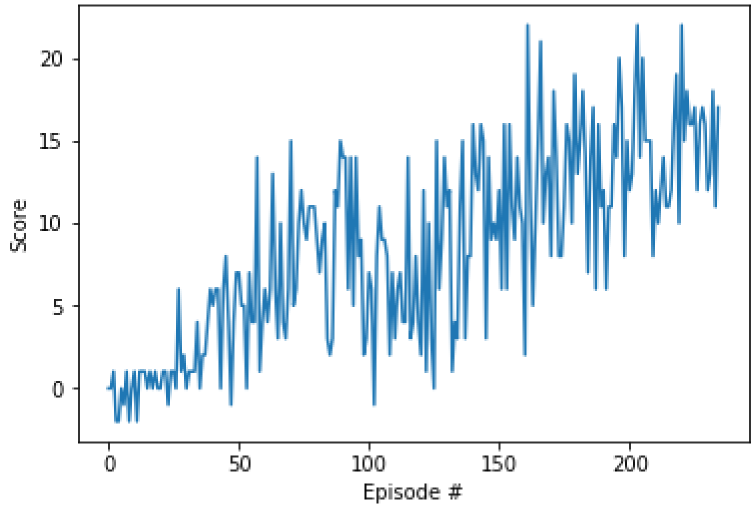
Source: https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4

Hyperparameters:

- Maximum steps per episode: 1000
- Starting epsilon: 1.0
- Ending epsilon: 0.02
- Epsilon decay rate: 0.96

Best Result:

Episode 100 Average Score: 4.59
Episode 200 Average Score: 10.11
Episode 235 Average Score: 13.00
Environment solved in **235** episodes! Average Score: 13.00



Other Results:

In total, I have done 64 runs with different hyperparameters, algorithms (dueling and double DQN) and neural networks. My top 10 results use all the DQN algorithm with a 64 (fc1_units), 64 (fc2_units) neural network. The biggest impact had the change of the EPS decay rate from 0.999 to 0.96.

Top 10 Results:

A	B	C	D	E	F	G	H
Type	fc1_units	fc2_units	Episodes to solve environment	Average Score	Eps Start	Eps End	Eps Decay
DQN	64	64	235	13.00	1.0	0.02	0.96
DQN	64	64	249	13.01	1.0	0.04	0.90
DQN	64	64	271	13.01	1.0	0.04	0.96
DQN	64	64	277	13.02	1.0	0.02	0.90
DQN	64	64	279	13.06	1.0	0.01	0.90
DQN	64	64	286	13.01	1.0	0.01	0.96
DQN	64	64	288	13.04	1.0	0.03	0.96
DQN	64	64	301	13.05	1.0	0.04	0.85
DQN	64	64	302	13.06	1.0	0.05	0.90
DQN	64	64	308	13.07	1.0	0.04	0.98

All other results (with Double DQN, Dueling DQN, different convolutional layers)

https://docs.google.com/spreadsheets/d/1du2mcoiwSwLUT331zakxkW0hfP7S00GMonyVpMxSk8Q/edit?usp=sharing

Sources:

https://github.com/udacity/deep-reinforcement-learning/blob/master/p1_navigation/README.md
https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf