Navigation Report

Learning Algorithm

In this project I used an implementation of the basic *Deep-Q Learning with experience replay* algorithm. The pseudocode is described as following:

Algorithm:

1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights theta_1
3. Initialize target action-value function Q^ with weights theta_2 = theta_1
4. For episode = 1, M do
    - Reset environment and get state
    - For t 5 1,T do
        - With probability e select a random action at
        - otherwise select the action with the highest Q value
        - Execute action at and observe reward rt and next state
        - Store transition state, at, rt and next state as experience
        - Set state as next state for the next step
        - Sample random mini-batch from experience buffer
        - Set yj as rj if episode terminates at step j+1 otherwise rj plus gamma time the max Q value for the next state
        - Perform gradiente descent with respect to the network parameters theta_1
        - Every C steps reset Q^ = Q
        - End For

- End For

Hyperparameters:

- `episodes = 3_000` ; maximum number of training episodes
- `max_t = 1_000` ; maximum number of timesteps per episode
- `eps_start = 1.0` ; starting value of epsilon, for epsilon-greedy action selection
- `eps_end = 0.01` ; minimum value of epsilon
- `eps_decay = 0.995` ; multiplicative factor (per episode) for decreasing epsilon
- `score_window_size = 100` ; the window size to compute moving avg of scores
- `solved_at = 13.0` ; env cosidered solved, when the moving avg of scores reaches this threshold

Agent hyperparameters.

- `replay_buffer_size = 100_000` ; how many transitions to store for replay
- `batch_size = 64` ; mini-batch size for training the neural network
- `gamma = 0.99` ; discount factor.
- `tau = 10E-3` ; for soft update of target parameters.
- `lr = 5E-4` ; learning rate.
- `update_every = 4` ; frequency to update the network

Neural Network Architecture

A sequential model with the following structure: - Layer 1: feed forward with 37 inputs(state size) and 512 nodes with ReLU activations. - Layer 2: feed forward with 512 inputs and 512 nodes with ReLU activations. - Layer 3: feed forward with 512 inputs and 4(action size) nodes.

Plot of Rewards

This agent was able to solve the environment with the selected hyper parameter at 1971 episodes: Plot of rewards

Ideas for Future Work

In order to improve the agent's performance, future work will include: 1. Fine-tune hyper parameters, training longer for example. 2. Represent states as raw pixel, to do this, we need to change our neural network architecture to include CNNs. 3. Upgrade our dqn algorithm to Rainbow.