HW5 Report

* Explain why DQN algorithm need these function and how you implement them:
  + Experience replay
    - Reinforcement learning is known to be unstable when a nonlinear function approximator such as a neural network is used to represent the action-value (Q) function. One cause of this instability is the correlations present in the sequence of observations. The experience replay technique can randomize over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution.
    - Implementation
      * In the source code, there is a class “ReplayMemory” defined to store a list of memory: the information of the previous state, action, reward, the next state, and whether the game is over. For each piece of the information, there is a list defined to store them. For example, a list “actions” will be used to store the action for each memory, and etc. At each step of the training, a new memory will be added to a ReplayMemory object in the “Agent” class.
  + Target network
    - Another cause of instability of reinforcement learning with neural network is the correlations between the action-values (Q) and the target values. Using a separate network for generating the targets in the Q-learning update can avoid this issue.
    - Implementation
      * The target network is built in the “build\_dqn” function of the agent.py file. Basically, it is a clone of the main DQN network. For every C steps (C = 10000) we update the target network by the main network and use to generate the Q-learning targets y for the following C updates to .
  + Epsilon-greedy policy
    - An Epsilon-greedy policy follows the greedy policy with probability (1 – Epsilon) and selects a random action with probability Epsilon, and this ensures adequate exploration of the state space.
    - Implementation
      * The Epsilon-greedy policy is implemented in the “predict” function of the agent.py file. In the “predict” function, a random floating number in the range of 0.0 and 1.0 is generated and compared to an epsilon value to implement Epsilon-greedy policy.

Clip reward

* + - Since the scale of scores varies greatly from game to game, clipping reward in the range [-1 1] limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games. This also improves the performance of an agent since it cannot differentiate between rewards of different magnitude.
    - Implementation
      * The clip reward mechanism is implemented in the “observe” function of the agent.py file. Reward values will be clipped between -1 and 1 before being used to train the network.
* If a game can perform two actions at the same time, how will you solve this problem using DQN algorithm?
  + Add extra actions that are the combination of two actions to the existing action set. For example, if the original action set is S1 = [a1, a2, a3], we add new actions a12, a13 and a23 to the action set. The final action set will become S2= [a1, a2, a3, a12, a13, a23], and we use this set as the output of the DQN.