##### AGENDA

首先我們會先簡介如何實作，接著再分為Env與Agent介紹，最後講我們訓練的結果

##### 簡介

我們使用DDQN作為主要的演算法

有兩個獨立的ConvNet，分別為Qonline跟Qtarget，這兩者為獨立的optimal action-value function

這兩個獨立的value function會由TD Estimate與TD Target來得到

最後，本篇使用epsilon-greedy action

##### Enviroment

我們使用前面提到的gym\_super\_mario\_bros作為我們的env，接著用JoypadSpace將action space限制在只做往右及往右上跳的動作。

接著因為馬力歐遊戲不會因為背景顏色而影響做的action，我們將其背景轉為灰階，如此，每個state的array會由[240,256,3] →[240,256,1]。接著將其resize成[84,84,1]。

而skipFrame是從gym.Wrapper中繼承而來的，主要是在執行step時，因為連續的Frame並不會差異很多，可以透過skip n個連續的Frame來加速運算，且不會lose過多data。至於reward的部分則會被累加起來。

最後就是用FrameStack將四個state合起來當作一個觀測點，也就是每個state的array會轉成[4,84,84,1]

##### Agent

我們創造了一個class, named Mario, 其中包含了3個主要的部份

1. Act: according to the optimal action policy based on the current state
2. memory: experiences. Experience = (current state, current action, reward, next state), we can use the benefit of Experience Replay to avoid stuck in the local area.
3. Learn: find a better action policy over time

## Act

For any given state, an agent can choose to do the most optimal action (exploit) or a random action (explore).

Mario randomly explores with a chance of self.exploration\_rate; when he chooses to exploit, he relies on MarioNet (implemented in Learn section) to provide the most optimal action.

## Memory

Cache and Recall functions serve as Mario’s “memory” process.

cache(): Each time Mario performs an action, he stores the experience to his memory. His experience includes the current state, action performed, reward from the action, the next state, and whether the game is done.

recall(): Mario randomly samples a batch of experiences from his memory, and uses that to learn the game.

## Learn

Mario uses the DDQN algorithm(Double Deep Q-Network)，分別為Qonline跟Qtarget,

In our implementation, we share feature generator features across Qonline跟Qtarget.

*"""mini CNN structure*

*input -> (conv2d + relu) x 3 -> flatten -> (dense + relu) x 2 -> output*

*"""*

使用上式中的mini CNN架構，再使用兩種不同的value function, 一種是TD Estimation, 另一種是TD Target。