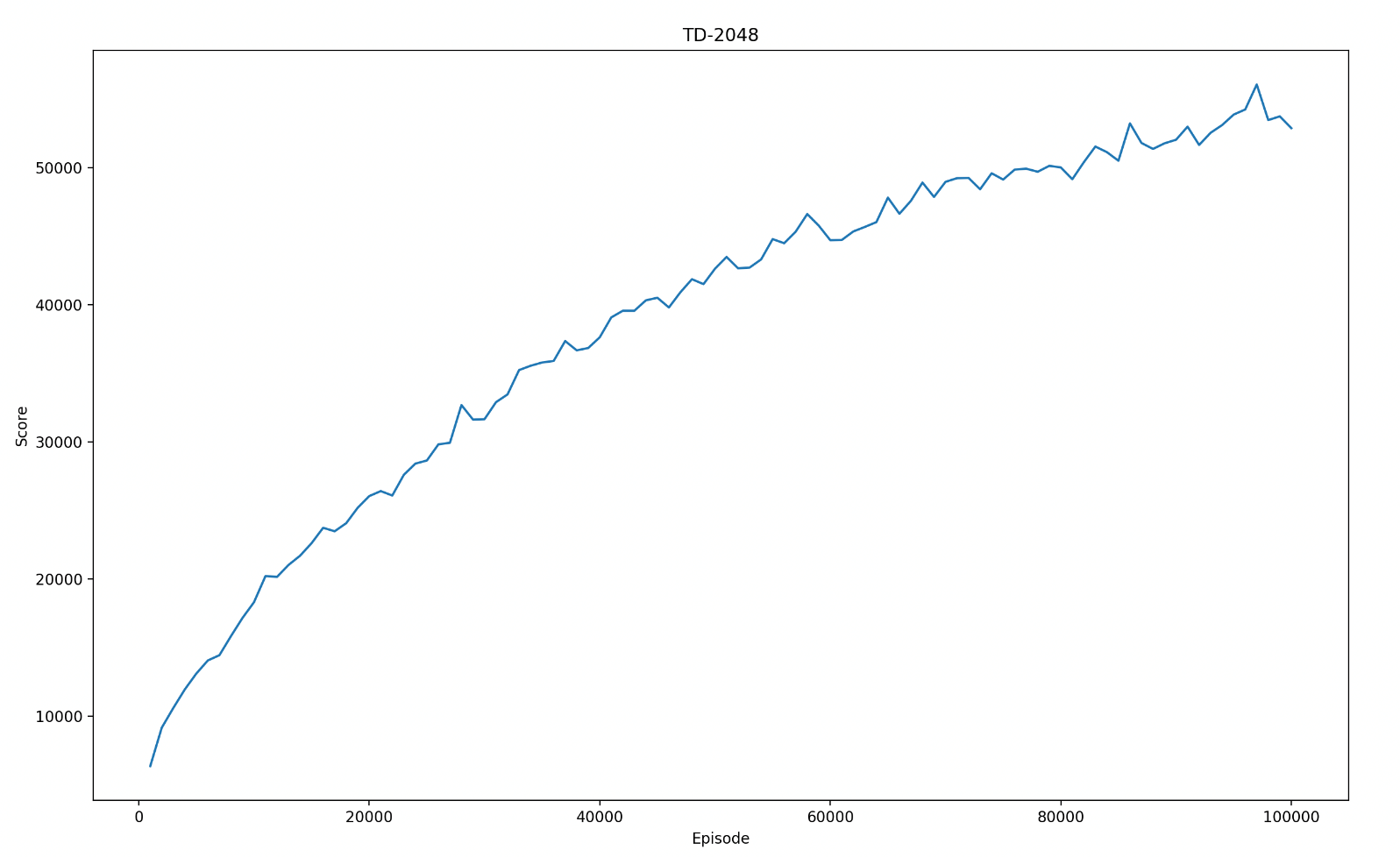
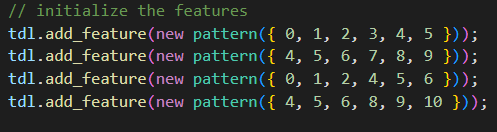
Experiment Report

* A plot shows scores (mean) of at least 100k training episodes



* Describe the implementation and the usage of 𝑛𝑛-tuple network.

我們可以透過n-tuple所算出來的值，來代表當前狀態的value。這樣除了可以節省記憶體，還可以加快運算。N-tuple是透過設幾個index是我們想觀察的值，將前面所選的一組index 以及對應的isomorphic，一共會有8個狀態，一起當作一個feature。那我們可能會有4個feature，我們在每次計算board value時，都會去計算每一個feature，以及裡面每一個isomorphic所算出來的value，做相加總後，當作是這個board的value。



在sample code已經訂好了 n-tuple的格式，我也沒有再特別更動。

* Explain the mechanism of TD(0).

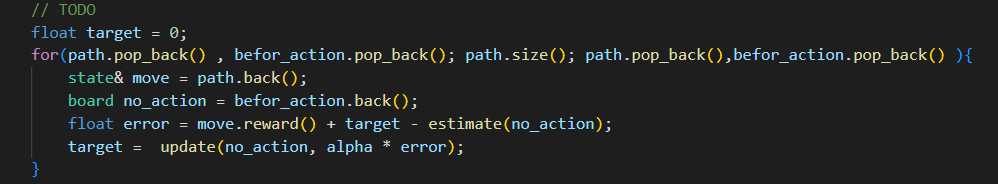
TD(0)是會根據我們選擇動作後，所產生的下一個狀態來去更新自己的weights 或者是 value。而蒙地卡羅是TD(1)，他是用遊戲結束的最後一個狀態的value，來去更新前面所有的狀態的weights。

* Describe your implementation in detail including action selection and TD-backup diagram.
  + Action Selection:



上面TODO我自己撰寫的程式碼。

1. 我們會先去針對每個動作算出他的reward值
2. 再針對action 後的board()，使用窮舉法，把每個空格出現2 或4的board()，全部算出來之後，取出他的value，乘上相對應的機率(2->0.9, 4->0.1)，全部加起來之後，再除以格子數。
3. 透過步驟2所算出的值，我們當作期望值，我們會拿reward+期望值，去更新目前board()還沒有action的value。最後選擇出哪一個action，所製造出來的value是最大的，我們就選擇哪個action。
   * TD-backup:



上面TODO我自己撰寫的程式碼。

要更新weight ，就是要用self value + learning rate \*( reward + next value – self value)。

我們透過recursive從尾巴(最後一個狀態)去不斷的更新weight。我把最後一個狀態的value當作是0，也就是遊戲結束的那個狀態，因為它是一個死路，所以將他設成0是為了讓model不要在進來這個會結束的狀態。而我們後面的next value是拿到已經更新好狀態的self value(這是上一輪更新的self value被當成這一輪的next value)，再來是self value 以及 reward，我們都可以拿到，就可以更新weights了。

至於code裡面的befroe\_action，是我把沒有經過select\_best\_move的board存起來，傳入這個funcction裡面，讓他當成我們self value的borad來源。