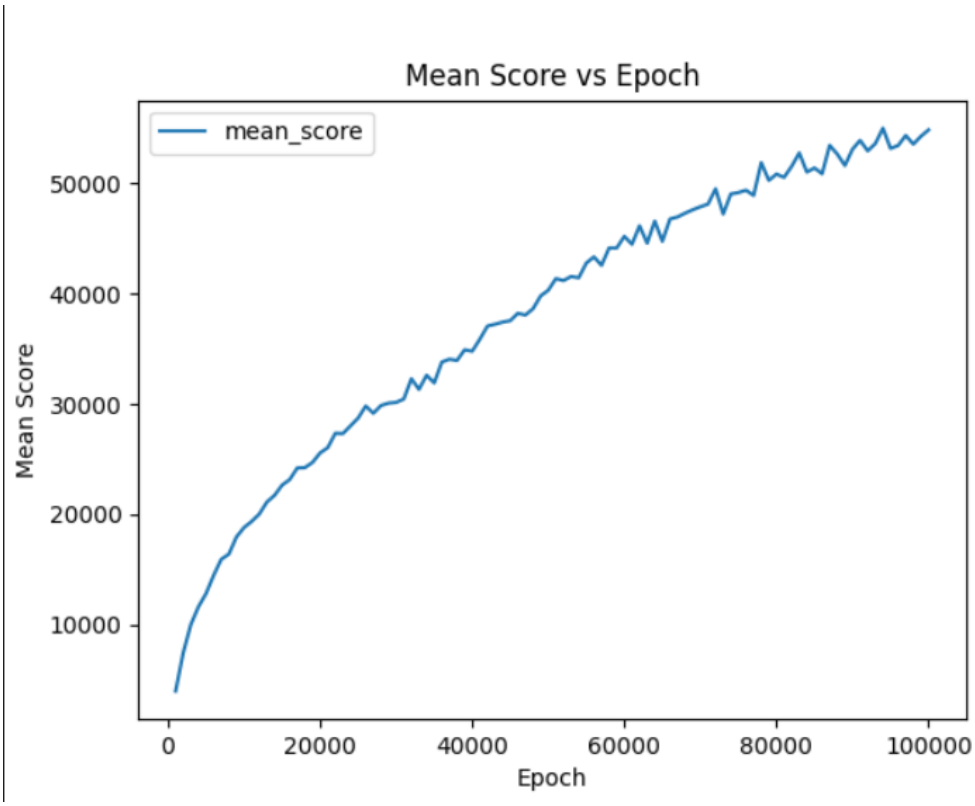
Lab2 report

學號：311512049 姓名：陳緯翰

Report (50%)

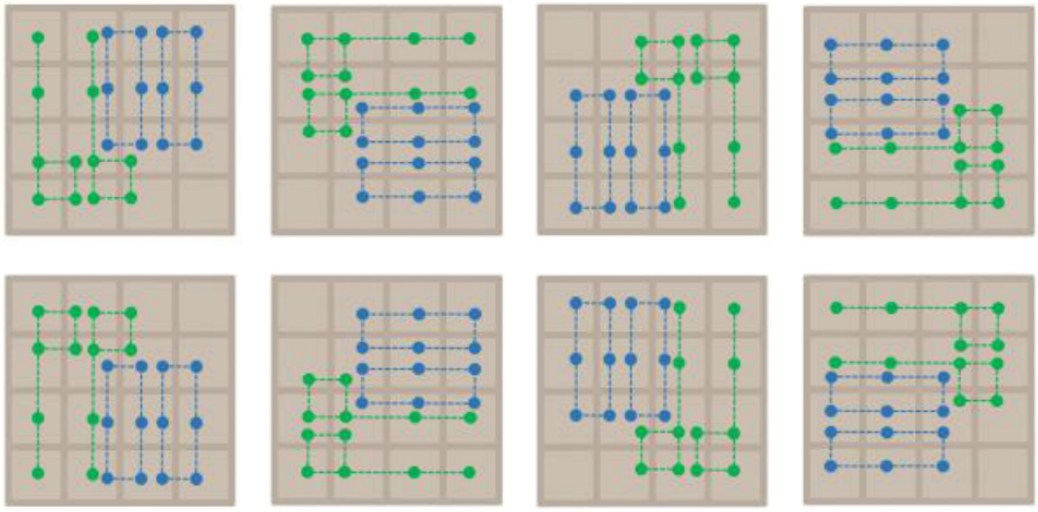
plot shows scores (mean) of at least 100k training episodes (10%)



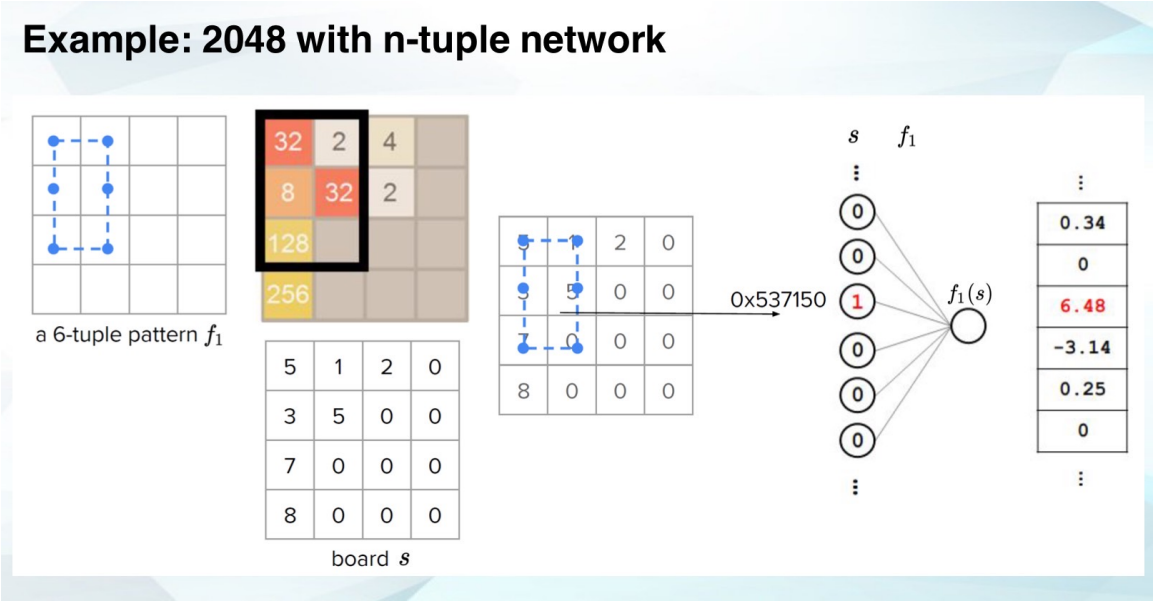
Describe the implementation and the usage of *nn*-tuple network. (10%)

- 因為有isomorphic，可將盤面設定成下面所式的8種不同樣式
- 4* 6 tuple network(有4個6 tuple 的network)

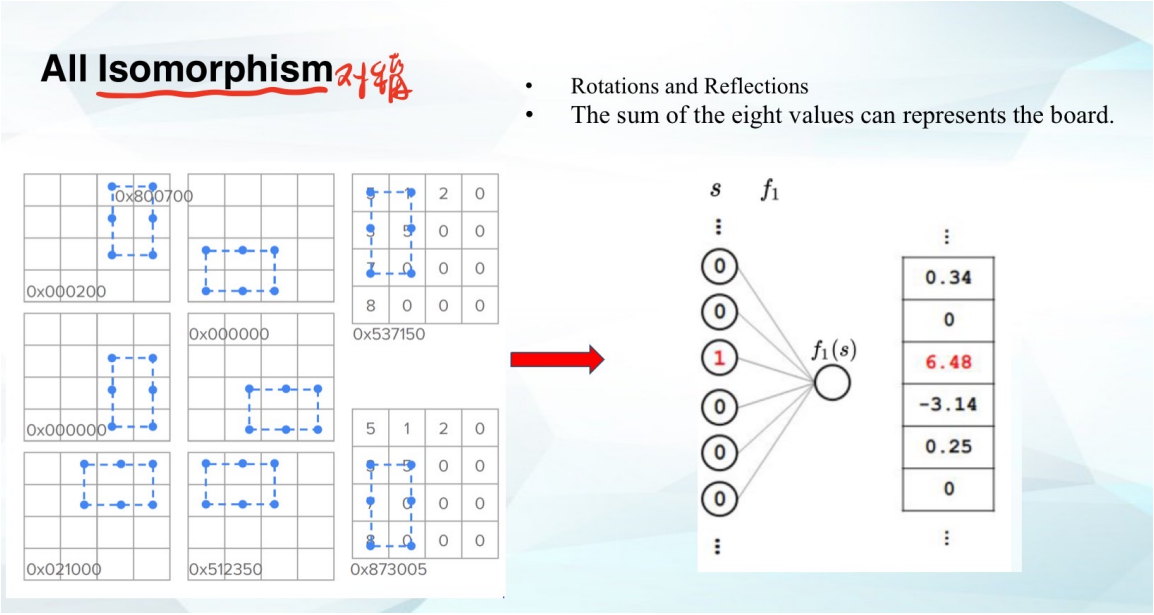
為何需要n-tuple network，因為若純粹考慮每一格的出現數字且將每一格都納入考慮會有size = $15^{16} * 4$ byte (2^0~ 2^{14})個資料需要考慮(資料量過大)，所以需要利用n-tuple， size = $4 * 15^6 * 4$ byte 來去進行有效執行來計算value。

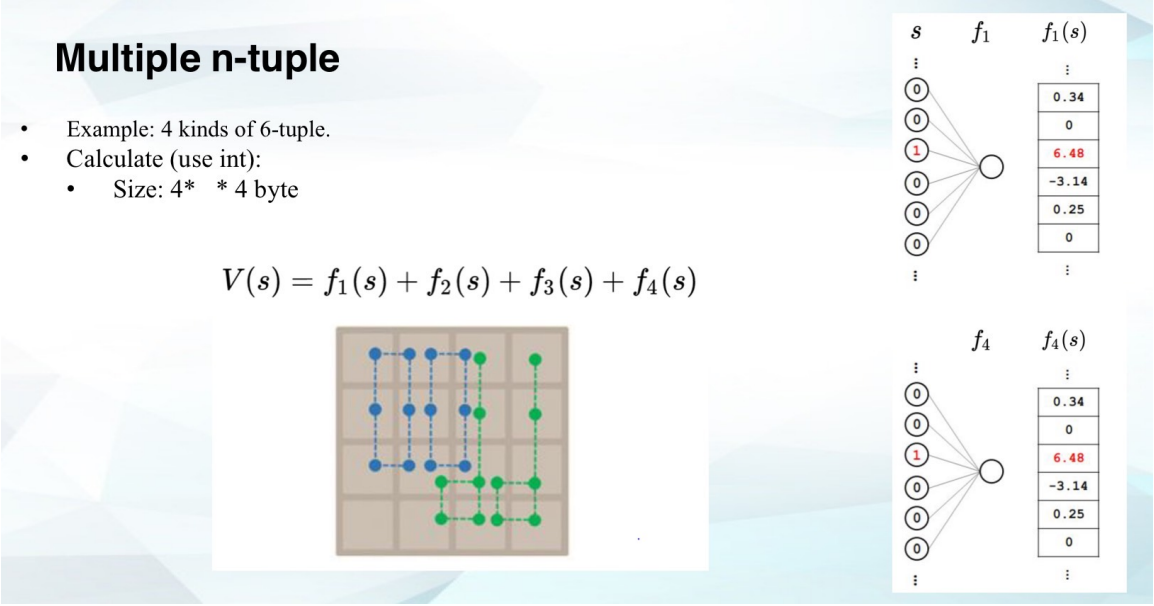


- 用index的方式將一個6格向量儲存到一個binary index，對應到1的會存取537150
- 在 n-tuple network 中，每個 n-tuple 都被映射到一個權重值，這些權重值被用來決定輸入資料所對應的輸出結果。對於一個新的輸入資料，它的特徵會被提取出來，然後和 n-tuple 的權重值進行匹配，最終得到輸出結果。n-tuple network 的訓練過程通常是通過梯度下降法來最小化預測結果和實際結果之間的差距。



- 用旋轉board的方式代表4*6tuple在下面的不同角落 (iso = 8)





Explain the mechanism of TD(0). (10%)

TD(0)是一種強化學習演算法，用於學習價值函數。它透過觀察代理與環境的互動，根據獎勵信號的反饋更新狀態的價值估計。

TD(0)的0代表更新狀態價值函數（state-value function）的方法只考慮當前狀態和下一個狀態之間的轉移，而不考慮從下一個狀態開始的未來時間步。

TD(0)演算法使用一種基於差異的更新策略，其中目標值是當前獎勵加上下一狀態的價值估計。透過計算當前狀態的價值估計和目標值之間的差異，可以得到更新所需的誤差。這個誤差可以用來更新當前狀態的價值估計。這種差異更新的方式被稱為「時差學習」（Temporal Difference learning）。

具體地，TD(0)演算法的更新規則如下：

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

其中，$V(S_t)$是在時間步t時狀態S_t的價值估計；R_{t+1}是在時間步$t+1$時代理收到的獎勵$V(S_{t+1})$是在時間步$t+1$時下一個狀態S_{t+1}的價值估計；α是學習率(set α = 0.1)，決定了每次更新的步長大小；γ是折扣因子(set γ = 1)，用於調整未來獎勵的重要性。

TD(0)演算法的主要優點是它只需要當前狀態和下一個狀態的價值估計就能進行更新，不需要對所有後續狀態進行遍歷，因此計算效率較高。它也可以在連續的狀態空間中進行操作，並且可以與其他強化學習演算法結合使用。

Describe your implementation in detail including action selection and TD-backup diagram. (20%)

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

更新V(s)estimate和update函數的實現都是遍歷一個預定義的 isom 數組中的所有變換，併計算每個變換在內部表格中的索引，然後將這些索引對應的值累加起來，得到最終的估值。 在learn_from_episode函數中，我們會調用這兩個函數來計算當前局面的估值，以及更新當前局面的估值。

這段代碼實現了一個評估函數 estimate，接受一個棋盤 b 作為參數，並返回一個浮點數，表示棋盤狀態的估值。該函數首先初始化一個名為 value 的浮點數變量，用於累計所有變換之後的值。然後遍歷一個預定義的 isom 數組中的所有變換，併計算每個變換在內部表格中的索引。這裡，indexof(isom[i], b) 函數返回一個整數值，表示將棋盤 b 進行變換 isom[i] 後的狀態在內部表格中的索引。最後，使用 operator[] 函數訪問內部表格，並將每個變換的估值相加到 value 變量中，以得到最終的估值。這個評估函數用於靜態評估棋盤狀態，以幫助搜索算法找到最佳下一步。

Algorithm:

A pseudocode of the game engine and training. (modified backward training method)

```
function PLAY GAME
  score ← 0
  s ← INITIALIZE GAME STATE
  while IS NOT TERMINAL STATE(s) do
    a ← argmax_{a'∈A(s)} EVALUATE(s, a')
    r, s', s'' ← MAKE MOVE(s, a)
    SAVE RECORD(s, a, r, s', s'')
    score ← score + r
    s ← s''
  for (s, a, r, s', s'') FROM TERMINAL DOWNTO INITIAL do
    LEARN EVALUATION(s, a, r, s', s'')
  return score

function MAKE MOVE(s, a)
  s', r ← COMPUTE AFTERSTATE(s, a)
  s'' ← ADD RANDOM TILE(s')
  return (r, s', s'')
```

TD-state

```
function EVALUATE(s, a)
  s', r ← COMPUTE AFTERSTATE(s, a)
  S'' ← ALL POSSIBLE NEXT STATES(s')
  return r + Σ_{s''∈S''} P(s, a, s'')V(s'')

function LEARN EVALUATION(s, a, r, s', s'')
  V(s) ← V(s) + α(r + V(s'') − V(s))
```

- estimation

```
/**
 * estimate the value of a given board
 */
virtual float estimate(const board& b) const {
    // TODO
    float value = 0;
    for (int i = 0; i < iso_last; i++) {    //遍历一
        size_t index = indexof(isomorphic[i], b);
        value += operator[](index);
    }
    return value;
}
```

- update

  update 會得出 V(s'') ，代表最佳盤面的value值

```
/**      henry890112, 2 weeks ago • lab2 first commit
 * update the value of a given board, and return its updated value
 */
virtual float update(const board& b, float u) {
    // TODO
    /*
    V(s) = V(s)+alpha(r + V(s'') - V(s)) --> V(s) = V(s)+ r + alpha* (V(s'') - V(s))
    --> V(s) = V(s)+ reward+ (alpha * error)
    更新V(s)
    */
    float adjust = u / iso_last;  //u = alpha * error
    float value = 0;
    for (int i = 0; i < iso_last; i++) {
        size_t index = indexof(isomorphic[i], b);   //indexof函数 (定義的)，遍歷所有的isom(共8個)
        operator[](index) += adjust;
        value += operator[](index);
    }
    return value;
}
```

- indexof

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
    // TODO
    /*
    << 是 C++ 中的左移位運算符。對於一個整數 a 和一個非負整數 b，a << b 表示將 a 的二進制
    表示向左移動 b 位，右邊補 0，然後將結果作為一個新的整數返回。
    在給定的代碼中，表達式 b.at(patt[i]) << (4 * i) 是將 b.at(patt[i]) 左移 4 * i 位。
    由於棋盤的每個位置可以有 16 種不同的狀態（空、X、O 其中之一），因此可以使用 4 個位元來
    存儲每個位置的狀態。因此，整個棋盤的狀態可以用一個長度為 4 * patt.size() 的二進制數字
    來表示，其中 patt.size() 是棋盤中可用的位置數量。由於 index 是用來存儲這個二進制數字的整數，
    所以在這裡使用左移位運算符來將每個位置的狀態放到正確的位置上，從而構造出整個棋盤的狀態對應的整數值。
    */

    size_t index = 0;
    for (size_t i = 0; i < patt.size(); i++) {
        index += b.at(patt[i]) << (4 * i);  //將棋盤狀態轉換成整數
    }
    return index;
}
```

- select_best_move

  走片board中16格將popup 2給ratio 0.9且popup 4給0.1去更新value的分數

```
state select_best_move(const board& b) const {
    state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
    state* best = after;
    for (state* move = after; move != after + 4; move++) {
        if (move->assign(b)) {
            // TODO
            //選出最好的環境state(對全部estimate的環境做平均)
            // 選擇最佳的環境state(b)

            board after_move_state = move->after_state();
            float estimate_value = 0;
            int num = 0;

            for(int i = 0; i < 16; i++){
                if(after_move_state.at(i)==0){
                    // popup 2 90%
                    after_move_state.set(i,1);
                    estimate_value += 0.9*estimate(after_move_state);
                    // popup 4 10%
                    after_move_state.set(i,2);
                    estimate_value += 0.1*estimate(after_move_state);
                    after_move_state.set(i,0);
                    num++;
                }
            }
            estimate_value = estimate_value/num;
            move->set_value(move->reward() + estimate_value);
            if (move->value() > best->value())
                best = move;
        } else {
            move->set_value(-std::numeric_limits<float>::max());
        }
        debug << "test " << *move;
    }
    return *best;
}
```

- update_episode

```cpp
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    //利用alpha來更新環境      henry890112, 2 weeks ago • finish the TODO …
    float target = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = target - estimate(move.before_state());
        target = move.reward() + update(move.before_state(), alpha * error);
        debug << "update error = " << error << " for" << std::endl << move.before_state();
    }
}
```

- Result

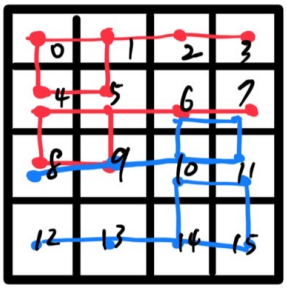
```
100000  mean = 54806.5  max = 161924
        64      100%    (0.2%)
        128     99.8%   (0.1%)
        256     99.7%   (1.1%)
        512     98.6%   (1.7%)
        1024    96.9%   (12%)
        2048    84.9%   (29%)
        4096    55.9%   (51.2%)
        8192    4.7%    (4.7%)
```

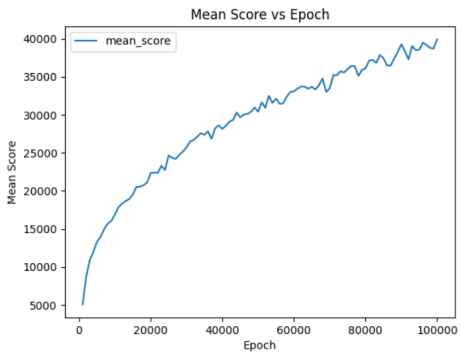
- my try

1. 更改n-tuple network

   因為想說每次訓練時若能將每個都納入考慮可能會獲得較好的結果，所以利用下圖的n-tuple方法來進行訓練，但結果並沒有比較好。

```
        8192    0.6%    (0.6%)
100000  mean = 39923    max = 124520
        64      100%    (0.1%)
        128     99.9%   (0.3%)
        256     99.6%   (0.5%)
        512     99.1%   (4.7%)
        1024    94.4%   (22.8%)
        2048    71.6%   (38.6%)
        4096    33%     (32.5%)
        8192    0.5%    (0.5%)
```



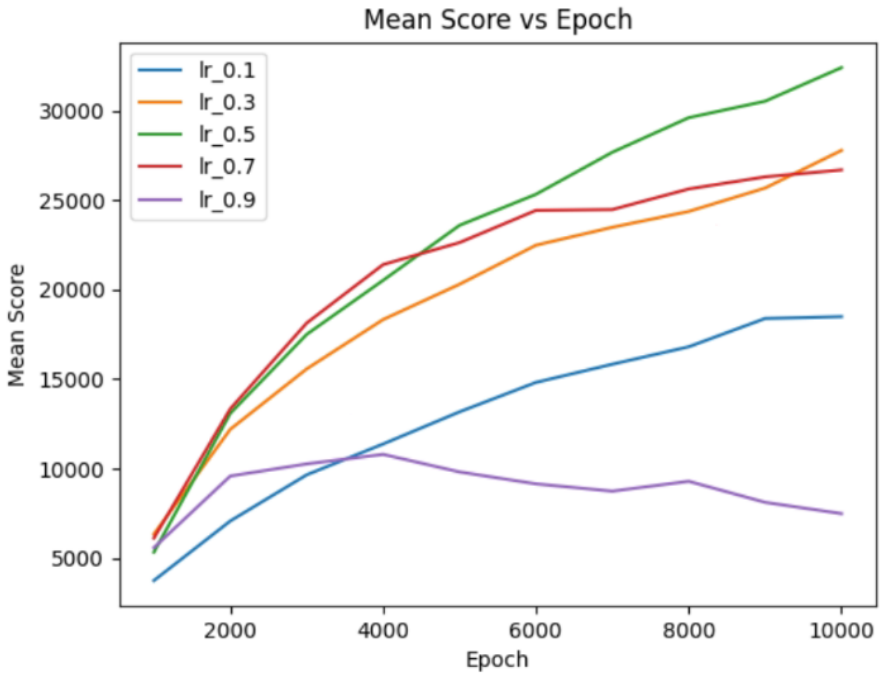


2. 調整learning rate

   在TD(0)中，learning rate決定了每一次更新中新觀測到的狀態和先前估計值之間的權衡。如果learning rate設置過高，則每一次更新會對之前的估計值產生較大的影響，可能會導致算法發生不穩定或不收斂的情況。相反，如果learning rate設置過低，則算法收斂速度較慢，需要更長的時間才能達到最優解。

   因此，調整TD(0)中的learning rate的目的是找到一個最適合的值，可以使算法在可接受的時間內收斂到最優解，並且避免出現不穩定的情況。透過調整learning rate，可以幫助算法更好地利用過去的經驗來進行更新，從而提高算法的性能。在實際應用中，可以進行多次實驗，選擇最佳的learning rate值，以取得最好的效果。

   下圖為我利用不同learning rate在epoch = 10000所產生出來的結果(其他皆沒有做更改)，由圖可見learning rate 在0.5會有較好的表現



- learning rate = 0.1

```
10000   mean = 18492.3   max = 49852
        64      100%     (0.2%)
        128     99.8%    (0.5%)
        256     99.3%    (2.5%)
        512     96.8%    (16.2%)
        1024    80.6%    (56%)
        2048    24.6%    (24.3%)
        4096    0.3%     (0.3%)
```

- learning rate = 0.3

```
10000   mean = 27778.1   max = 76368
        64      100%     (0.1%)
        128     99.9%    (0.4%)
        256     99.5%    (2.5%)
        512     97%      (7.3%)
        1024    89.7%    (30.9%)
        2048    58.8%    (50.1%)
        4096    8.7%     (8.7%)
```

- learning rate = 0.5

```
10000   mean = 32402.2   max = 84124
        64      100%     (0.1%)
        128     99.9%    (0.3%)
        256     99.6%    (1%)
        512     98.6%    (7.2%)
        1024    91.4%    (20.8%)
        2048    70.6%    (52.8%)
        4096    17.8%    (17.8%)
```

- learning rate = 0.7

```
10000   mean = 26686    max = 77984
        32      100%    (0.1%)
        64      99.9%   (0.3%)
        128     99.6%   (1.4%)
        256     98.2%   (1.3%)
        512     96.9%   (8.9%)
        1024    88%     (25.3%)
        2048    62.7%   (52.8%)
        4096    9.9%    (9.9%)
```

- learning rate = 0.9

```
10000   mean = 7488.15   max = 15828
        32      100%     (0.5%)
        64      99.5%    (0.8%)
        128     98.7%    (2.5%)
        256     96.2%    (12.9%)
        512     83.3%    (50.2%)
        1024    33.1%    (33.1%)
```

```cpp
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    //利用alpha來更新環境      henry890112, 2 weeks ago • finish the TODO …
    float target = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = target - estimate(move.before_state());
        target = move.reward() + update(move.before_state(), alpha * error);
        debug << "update error = " << error << " for" << std::endl << move.before_state();
    }
}
```