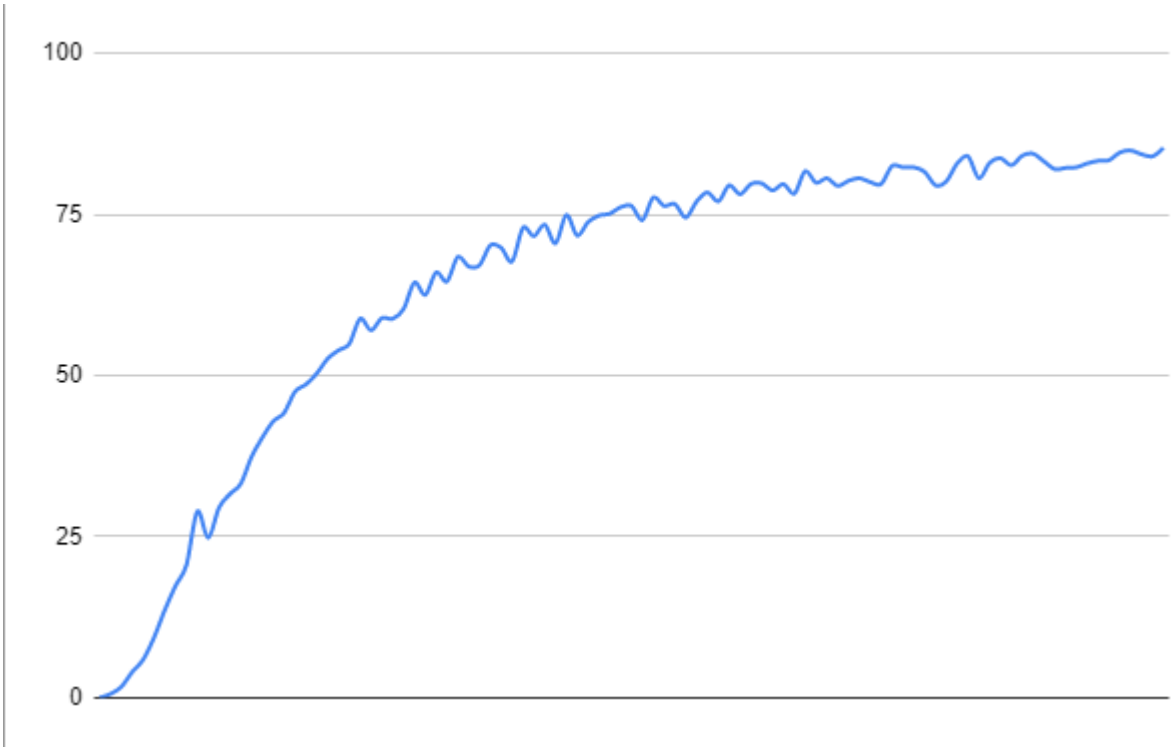
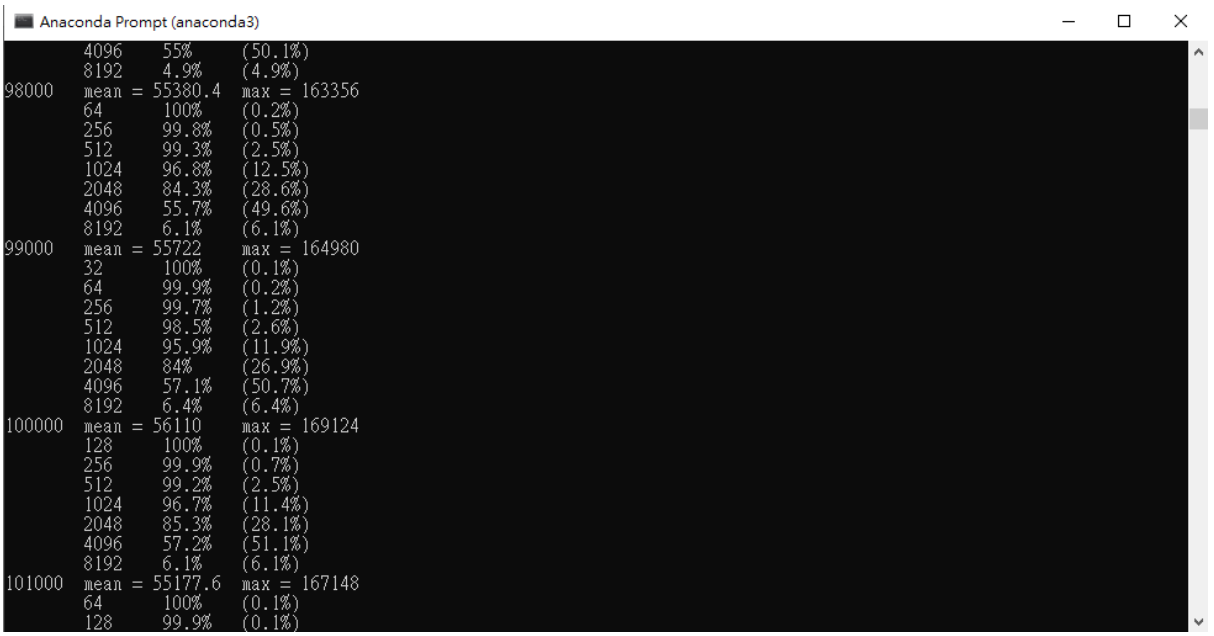
DLP_LAB2_309553012_黃建洲

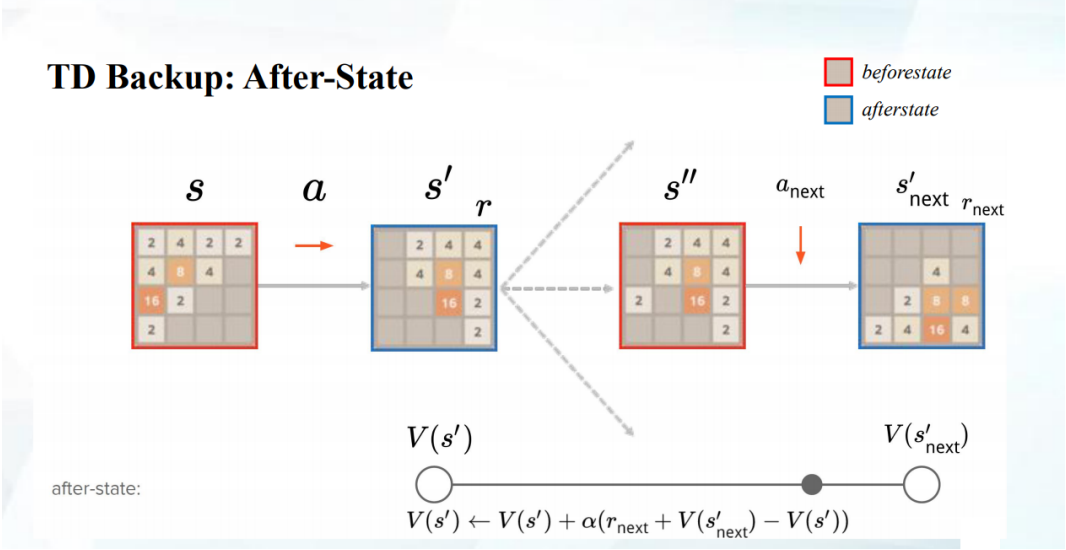
1. A plot shows episode scores of at least 100,000 training episodes





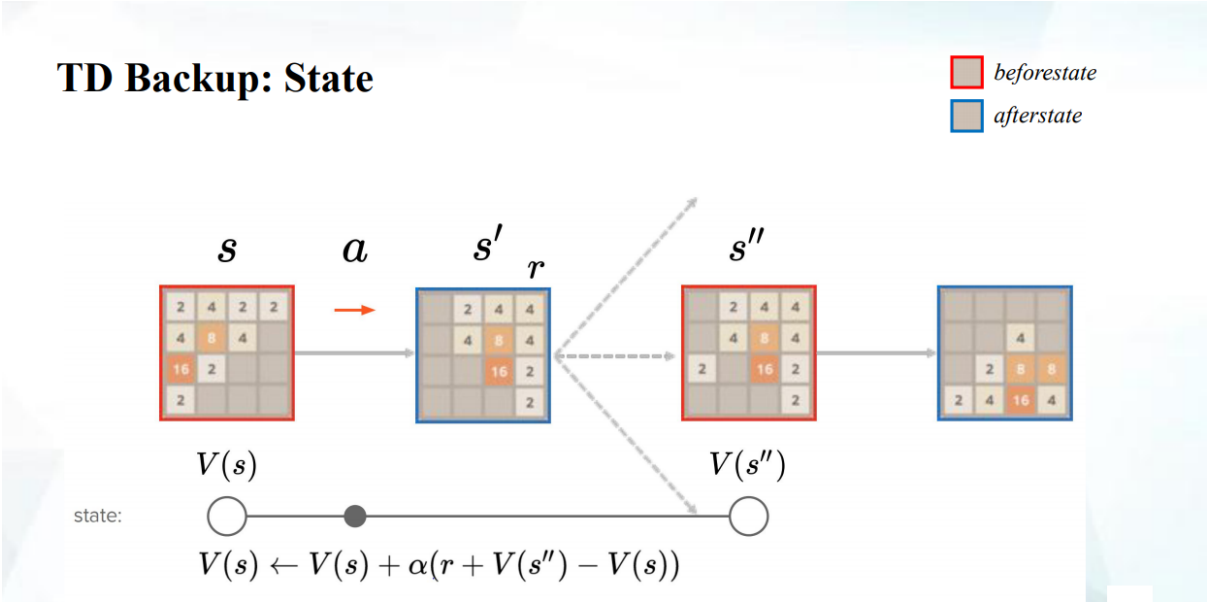
2. Describe the implementation and the usage of n-tuple network.
由於2048遊戲中，若以最大值為32768為例子的話，每個格子中都有15種可能性，則state就會有15^16種，Q表會因為太大而建不起來。
n-tuple network是用來解決這個問題的一個方法，改為使幾個格子內的數值為一個feature，以feature為單位來表示當前的state，每個feature自己計算出一個feature value用來代表觀察值。每一個feature還會再經過盤面的旋轉與鏡像處理，產生出八個同構feature，最後把所有特徵的分數加起來代表整個盤面的分數。

3. Explain the mechanism of TD(0)
   TD(0)，也是Q-learning的一種，把遊戲進程分成好幾個state，以2048來說就是在盤面更新後(before-state)到玩家做出決策後更新的盤面之前(after-state)。玩家在每個state會根據Q-value選擇出一個action進行實作，並評估出一個reward值。最後根據這些結果來更新在更新回Q-value中。
4. Explain the TD-backup diagram of V(after-state)



   在開始進行更新時，將end state的value設為0，並從倒數第二個state開始進行迭代。每次迭代時更新的是after-state的value，根據公式再將learning rate乘上next state reward加上next state value減掉這個state的value。每次迭代的意義都在將之後的state的影響代入目前這個state。

5. Explain the action selection of V(after-state) in a diagram
   After state的部分在選擇動作時使用的方式比較簡單，純粹比較每個action造成的after state value，使用最好的action進行更新。
6. Explain the TD-backup diagram of V(state)



與after state的差異在於，迭代的所更改的是before state而不是after state的value。並且next state的評估值在指派的時候也會有所差異。在after state中是reward與next state的after state盤面值。而在state中則是before state之盤面值。

7. Explain the action selection of V(state) in a diagram
state的部分在選擇動作時會在往後看一個階段，當每一個action套用並移動到after state後，會再計算所有可能的下一個before state，並計算value與出現這個before state的機率的乘積，最後將所有的相加得到該action的值。
8. Describe your implementation in detail
由於助教有給after state版本的範本，並且在state版本中我們只需要填入5個function。前面三個estimate, update和indexof的function因為只是算法，可以直接從after state的版本中沿用。其他兩個也大部分都可以進行沿用。
有進行更改的部分:
1. select best action:

在after state的版本中，僅僅只是分別比較4個action所產生的after state之盤面評估值與reward之和，並選用分數最高的action作為這次的output。Before state的做法如下圖，在每一個action中要考慮到所有可能的next before state。首先獲取目前盤面上尚沒有數字的格子index，在這些空格中有90%的機率產生2，以及10%的機率產生4。最後求所有衍生的盤面值與機率相乘後之和，比較後選出最佳的action。Set value的部分由於是要將值存給episode update用，所以根據公式改存入該狀態下before state的盤面值。

```
for (state* move = after; move != after + 4; move++) {
    if (move->assign(b)) {
        // TODO
        int space[16], num = 0;
        float rw = (float)move->reward();

        for (int i = 0; i < 16; i++){
            if (move->after_state().at(i) == 0) {
                space[num++] = i;
            }
        }
        if(num){
            for(int i = 0; i<num; i++){
                board* tmp_b_2 = new board(uint64_t(move->after_state()));
                board* tmp_b_4 = new board(uint64_t(move->after_state()));
                tmp_b_2->set(space[i],1);
                tmp_b_4->set(space[i],2);
                rw += estimate(*tmp_b_2) * 0.9f / num;
                rw += estimate(*tmp_b_4) * 0.1f / num;
                delete tmp_b_2;
                delete tmp_b_4;
            }

            move->set_value(estimate(move->before_state()));
        }
        if(rw > rw_max){
            best = move;
            rw_max = rw;
        }
```

2. episode update

除了在selection best move中提到要改set value的值外，唯一的改動就是在計算跌代的next state value時，在update function裡面要用before state來取代after state而已。如下圖:

```cpp
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float exact = 0;
    for (path.pop_back() /* terminal state */; path.size(); path.pop_back()) {
        state& move = path.back();
        float error = exact - (move.value() - move.reward());
        debug << "update error = " << error << " for after state" << std::endl << move.after_state();
        exact = move.reward() + update(move.before_state(), alpha * error);
    }
}
```

9. Best Result:
   Episode = 983000
   2048 win-rate = 93.5%