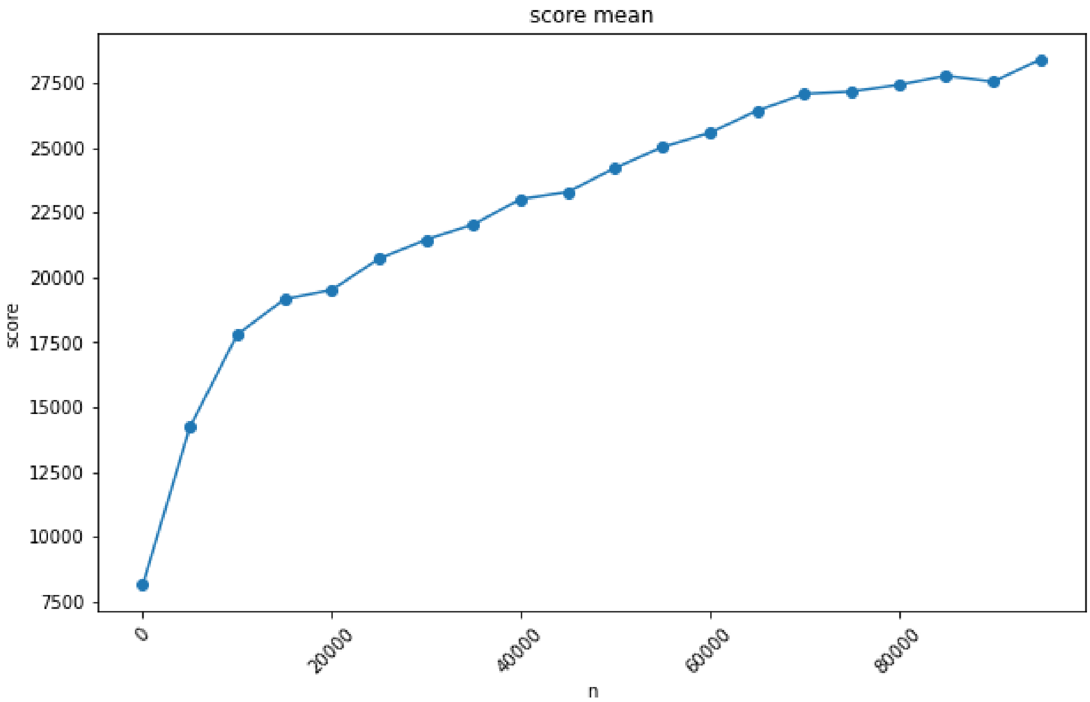
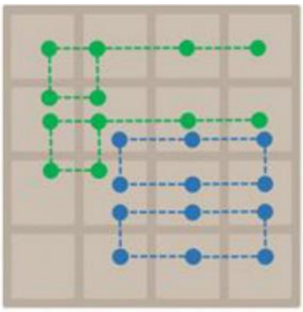
311706009 資管碩二 王廣和

Performance:



Bonus:

- Describe the implementation and the usage of n-tuple network. (5%)
  n-tuple network 類似 cnn 取局部特徵，因為我們無法窮舉16^{16}所有可能性，所以改以16^{n}來代替，以 sample code 裡面就是取 6 位數 x 4 feature

  盤面是可以 rotate 與 mirror 的，因此有 isomorphic(共構?)處理，一個 feature 會有 8 個相同性質，在計算 error 的時候也是 8 組 feature weight pair 做更新
- Explain the mechanism of TD(0). (5%)
  TD 跟 Q-learning 不同，value function 只針對 state 做評分
  - Update V(state), not V(after-state).
    - That is, you need to use the information of P(popup tile 2) = 0.9 and P(popup tile 4) = 0.1 in your code.

$$\textbf{function } \text{LEARN EVALUATION}(s, a, r, s', s'')$$
$$V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$$

由這兩段我寫成如下:

```
void update_episode(std::vector<state>& path, float alpha = 0.1) const {
    // TODO
    float td_target = 0;
    // path是從頭到尾的操作sequence
    // pop_back從最後一個開始拿，會改變vector；back不會改變，純取不刪

    for (path.pop_back(); path.size(); path.pop_back()){
        state& move = path.back();
        // 不是很理解這個requirement: Update V(state), not V(after-state)
        // t = t 的after state都還沒有考慮進popup，所以要使用 t+1 的before_state

        /**
         * s = before state， s' = after state，s'' = t+1 's before state
         * V(s) = V(s) + alpha (reward + V(s'') - V(s))
         *
         */
        // td error = r_{t+1} + gamma * V(s_{t+1}) - V(s_t)
        // td error = td target - V(s_t)
        float td_error = td_target - estimate(move.before_state());

        // update return回來的已經是value function調整過後的
        // td target = r_{t+1} + gamma * V(s_{t+1})
        td_target = move.reward() + update(move.before_state(), alpha * td_error);
    }
```

我想說如果是考慮完 popup 後的盤面，應該都要取每個 state 的 before board 來計算($s_{t+1} - s_t$)，如果是 Q-learning 看 action 則用 after($s'_{t+1} - s'_t$)，但後者的表現似乎比前者好...可能是哪裡寫錯，不然 performance 有相當差距。

- Describe your implementation in detail including action selection and TD-backup diagram. (10%)

一個 feature 的 weight 在算時要考量共構的

```
virtual float estimate(const board& b) const {
    // TODO
    float value = 0;
    for (int i = 0; i < iso_last; i++)
    {
        size_t table_key = indexof(isomorphic[i], b);

        // operator[]已經被overide，可以輸入table key而找到對應的weight

        value += operator[](table_key);
    }
    return value;
}
```

```
/**
 * update the value of a given board, and return its updated value
 * u代表 alpha*error，算出來的error要除8，表示feature與他的共構小夥伴共8個
 */
virtual float update(const board& b, float u) {
    // TODO
    float adjust = u / iso_last;
    float value = 0;
    for (int i = 0; i < iso_last; i++)
    {
        // size_t是unsigned int
        size_t table_key = indexof(isomorphic[i], b);
        // update weight
        operator[](table_key) += adjust;

        //return esti value that the weight is changed
        value += operator[](table_key);
    }
    return value;
}
```

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
    // TODO
    // 從000000~ffffff都有對應一組weight，所以會有key value pair(table for lookup)來記錄，這裡就是input為board與board idx 而可以output對應的key
    size_t table_key = 0;
    /**
     *  input
     *  b = 0x 4312 7521 8653 2731
     *  patt = {0, 1, 2, 3, 4, 5}
     */
    for (size_t i = 0; i < patt.size(); i++)
        table_key |= b.at(patt[i]) << (4 * i);
    return table_key;
    // 0x000000
    // 0x000001
    // 0x000031
    // 0x000731
    // 0x002731
    // 0x032731
    // 0x532731
}
```

這裡其實就是 argmax_a，用 assign 來檢查動作是否 legal

```
state select_best_move(const board& b) const {
    state after[4] = { 0, 1, 2, 3 }; // up, right, down, left
    // initialize 4 state obj for constructor input(opcode = 0,1,2,3)代表上下左右四個動作的state
    //每個state會有 board: before_state, after_state，before就是當前board，after存做完動作後的board
    state* best = after;
    for (state* move = after; move != after + 4; move++) {
        //檢查上下左右的move是否為legal，亦即score是否不為-1，
        //assing就可以set before state跟after state兩個board 與 score
        if (move->assign(b)) {
            // TODO

            // score跟esti分別代表遊戲給的reward R_t與模型如何評估盤面好壞的V(S_t)
            // move->set_value(move->reward() + estimate(move->after_state()));


            //TODO 考慮popup的element 2(0.9) 4(0.1)
            std::vector<board> all_possible_afterstate;
            std::vector<float> all_possible_afterstate_prob;
            std::vector<int> empty_idx;
```

assign 完之後的 afterstate 還未考量到 popup 後的所有可能，需要窮舉所有可能會長 2、4 的位置，以 empty_idx 紀錄

```cpp
for(int i = 0; i < 16 ; i++){
    if (move->after_state().at(i) == 0)
    {
        empty_idx.push_back(i);
    }
}

for(int idx: empty_idx){
    board tmp = move->after_state();
    tmp.set(idx, 2);
    all_possible_afterstate.push_back(tmp);
    all_possible_afterstate_prob.push_back(0.9);
}
for (int idx : empty_idx)
{
    board tmp = move->after_state();
    tmp.set(idx, 4);
    all_possible_afterstate.push_back(tmp);
    all_possible_afterstate_prob.push_back(0.1);
}
```

嘗試做這段 pseudo code

function EVALUATE(s, a)

$s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)

$S'' \leftarrow$ ALL POSSIBLE NEXT STATES(s')

return $r + \Sigma_{s'' \in S''} P(s, a, s'') V(s'')$

```cpp
    float sum_value = 0;
    for (int i = 0; i < all_possible_afterstate.size(); i++)
    {
        // 2's weight: 0.9/13 ; 4's weight:0.1/13
        //std::cout << "board size:" << all_possible_afterstate.size() << std::endl;
        sum_value += estimate(all_possible_afterstate[i]) * (all_possible_afterstate_prob[i] / (all_possible_afterstate.size()/2));
    }
    //std::cout << "sum value:" << sum_value << std::endl;
    // r + sum_s''in S'' P(s,a,s'')V(s'')
    move->set_value(move->reward() + sum_value);

    if (move->value() > best->value())
        best = move;
} else {
    move->set_value(-std::numeric_limits<float>::max());
}
debug << "test " << *move;
```

311706009 資管碩二 王廣和

筆記

int64 代表一個 board 的盤面

```
 * 64-bit bitboard implementation for 2048
 *
 * index:
 *  0  1  2  3
 *  4  5  6  7
 *  8  9 10 11
 * 12 13 14 15
 * note that the 64-bit value is little endian
 * therefore a board with raw value 0x 4312 7521 8653 2731 ull would be
 * +------------------------+
 * |     2     8   128     4|
 * |     8    32    64   256|
 * |     2     4    32   128|
 * |     4     2     8    16|
 * +------------------------+
```

ull 是 unsigned long long

4312752186532731 的解讀方式是由左往右，所以 index 其實是 15, 14, 13, …,1 , 0

```
/**
     * get a 16-bit row
     */
    int  fetch(int i) const { return ((raw >> (i << 4)) & 0xffff); }
```

0 <= i <=3

先往左 i*4 格之後，再取最後四位

4312 7521 8653 2731 如果我要取 i = 0

raw >> (0 << 4)，0 << 4 就還是 0，即 raw >> 0 還是 raw，然後和 0xffff 做 and

4312 7521 8653 2731

0000 0000 0000 f f f f

0000 0000 0000 2731

```
/**
     * set a 16-bit row
     */
    void place(int i, int r) { raw = (raw & ~(0xffffULL << (i << 4))) | (uint64_t(r & 0xffff) << (i << 4)); }
```

311706009 資管碩二 王廣和

除了要 set 的那行以外其他不變 所以 取 or raw = raw|new

```
/**
     * get a 4-bit tile
     */
    int  at(int i) const { return (raw >> (i << 2)) & 0x0f; }
```

因為每個 tile 是 4 個 bit 所以是 << 2 之後取 0x0f(0x0000 0000 0000 1111)

0 << 2 = 0

1 << 2 = 4

2 << 2 = 8

3 << 2 = 12

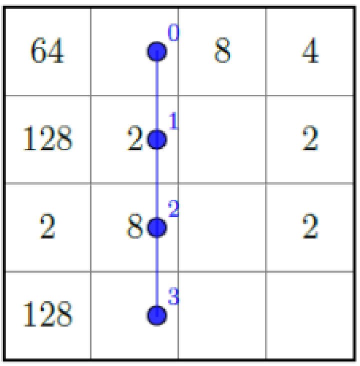
Isomorphic 共構，為了處理 rotate 與 mirror

```
/**
         * isomorphic patterns can be calculated by board
         *
         * take pattern { 0, 1, 2, 3 } as an example
         * apply the pattern to the original board (left), we will get
0x1372
         * if we apply the pattern to the clockwise rotated board
(right), we will get 0x2131,
         * which is the same as applying pattern { 12, 8, 4, 0 } to the
original board
         * { 0, 1, 2, 3 } and { 12, 8, 4, 0 } are isomorphic patterns
         * +-------------------------+       +-------------------------+
         * |     2     8    128     4|       |     4     2     8     2|
         * |     8    32     64   256|       |     2     4    32     8|
         * |     2     4     32   128| ----> |     8    32    64   128|
         * |     4     2      8    16|       |    16   128   256     4|
         * +-------------------------+       +-------------------------+
         *
         * therefore if we make a board whose value is
0xfedcba9876543210ull (the same as index)
         * we would be able to use the above method to calculate its 8
isomorphisms
         *
         * 簡言之就是一個 feature 會有 8 個同性質的，這邊就一併處理
         * 如果我想抓 pattern { 0, 1, 2, 3 }，但實際上{ 12, 8, 4, 0 }就是旋
轉之後的{ 0, 1, 2, 3 }
```

```
 * 旋轉 4 個 mirror 四個 一共 8 個
 */
for (int i = 0; i < 8; i++) {
    board idx = 0xfedcba9876543210ull;
    //處理 mirror
    if (i >= 4) idx.mirror();
    //處理 rotate
    idx.rotate(i);
    // p =  pattern({ 0, 1, 2, 3, 4, 5 })
    for (int t : p) {
        isomorphic[i].push_back(idx.at(t));
        // { 0, 1, 2, 3, 4, 5 }
        // { 3, 7, 11, 15, 2, 6}
        // { 15, 14, 13, 12, 10, 11}
        // { 0, 4, 8, 12, 9, 12}
        // 懶得列 mirror 了
    }
}
```

indexof

找到 feature 對應的 index，也就是這件事



0123	weight
0000	3.04
0001	−3.90
0002	−2.14
⋮	⋮
0010	5.89
⋮	⋮
0130	-2.01
⋮	⋮

```
size_t indexof(const std::vector<int>& patt, const board& b) const {
        // TODO
        // 從 000000~ffffff 都有對應一組 weight，所以會有 key value
pair(table for lookup)來記錄，這裡就是 input 為 board 與 board idx 而可以
output 對應的 key
        size_t table_key = 0;
        /**
         *  input
         *  b = 0x 4312 7521 8653 2731
         *  patt = {0, 1, 2, 3, 4, 5}
         */
        for (size_t i = 0; i < patt.size(); i++)
            table_key |= b.at(patt[i]) << (4 * i);
        return table_key;
        // 0x000000
        // 0x000001
        // 0x000031
        // 0x000731
        // 0x002731
        // 0x032731
        // 0x532731
    }
```

311706009 資管碩二 王廣和

estimate

```
/**
     * 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++)
        {
            size_t index = indexof(isomorphic[i], b);

            // operator[]已經被overide，可以輸入table key而找到對應的
weight
            // 把board上有出現的feature對應的weight sum 起來
            value += operator[](index);
        }
        return value;
    }
```