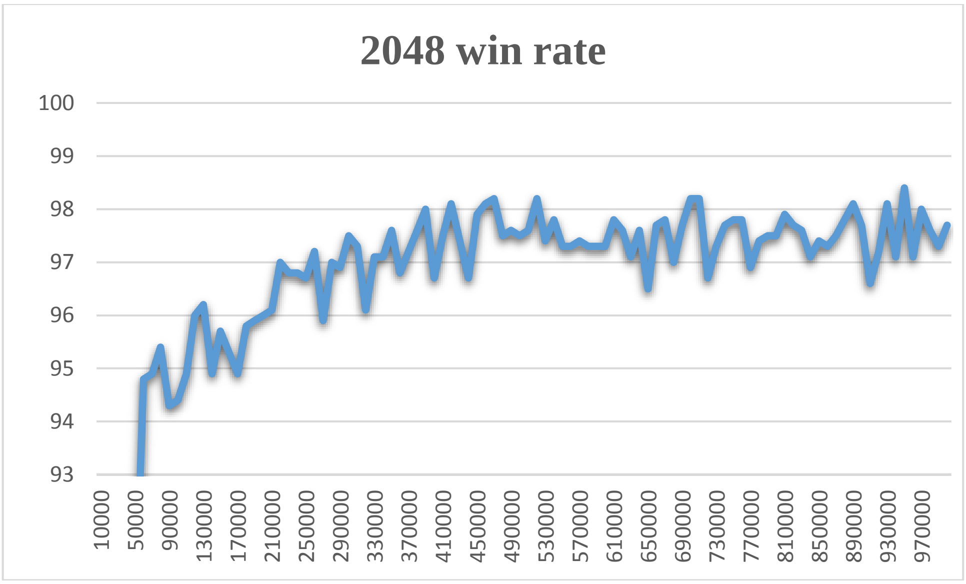
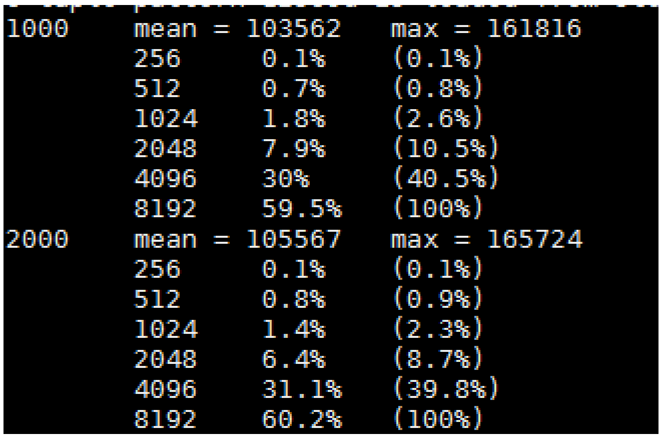
Deep Learning and Practice
Lab 8: Temporal Difference Learning

李韡
0556157
fm.bigballon@gmail.com

Win rate of 2048

I got the 97.5~98% win rate of 2048. The following figures show the win rate's changes in 100W iterations.



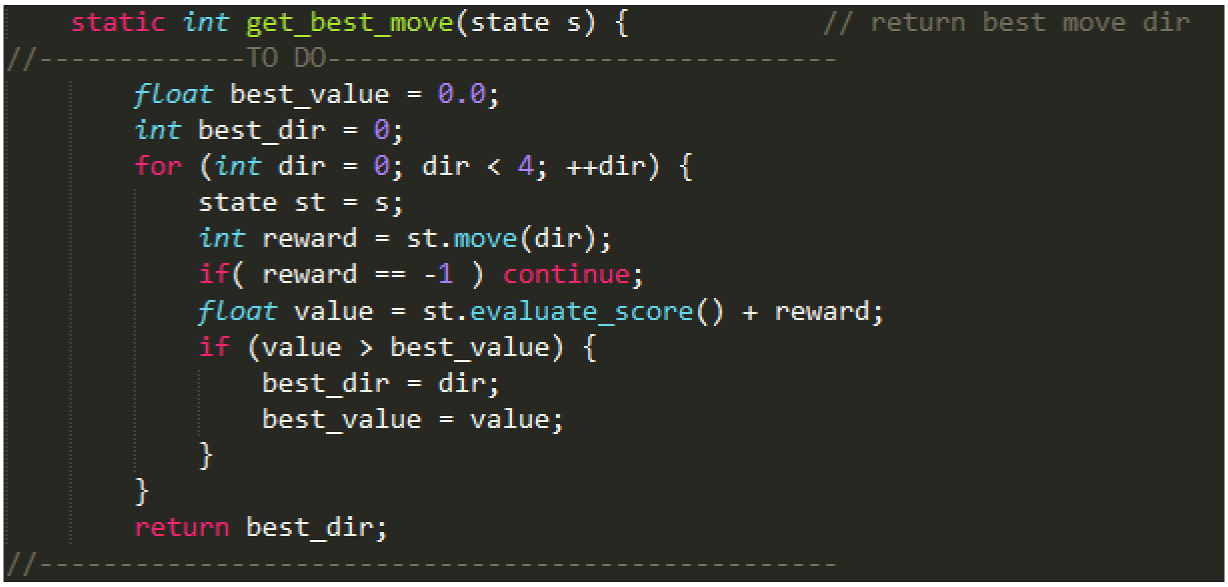


```
1000    mean = 103562   max = 161816
        256     0.1%    (0.1%)
        512     0.7%    (0.8%)
        1024    1.8%    (2.6%)
        2048    7.9%    (10.5%)
        4096    30%     (40.5%)
        8192    59.5%   (100%)
2000    mean = 105567   max = 165724
        256     0.1%    (0.1%)
        512     0.8%    (0.9%)
        1024    1.4%    (2.3%)
        2048    6.4%    (8.7%)
        4096    31.1%   (39.8%)
        8192    60.2%   (100%)
```

Figure 1: 2048 win rate

Report

■ Describe how you implement AI::get_best_move()



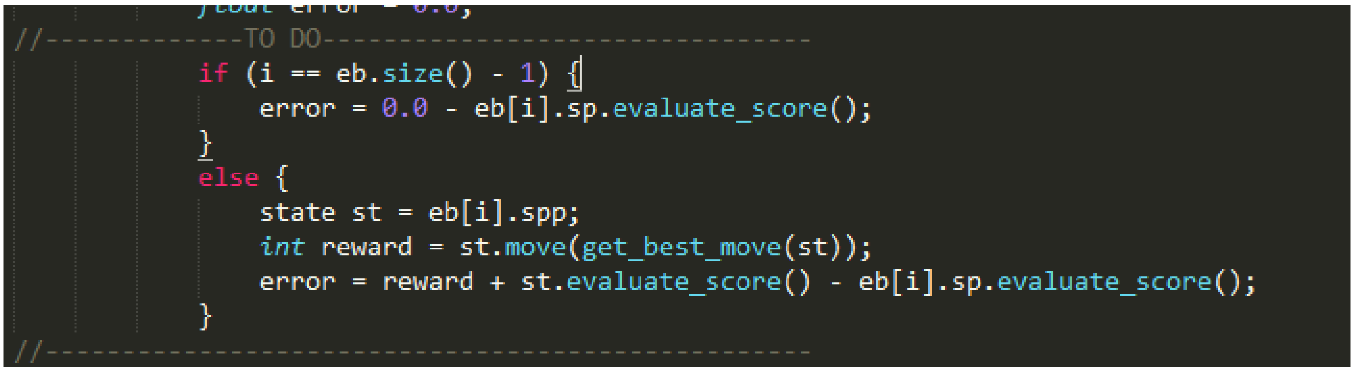
```
    static int get_best_move(state s) {           // return best move dir
//------------TO DO--------------------------------
        float best_value = 0.0;
        int best_dir = 0;
        for (int dir = 0; dir < 4; ++dir) {
            state st = s;
            int reward = st.move(dir);
            if( reward == -1 ) continue;
            float value = st.evaluate_score() + reward;
            if (value > best_value) {
                best_dir = dir;
                best_value = value;
            }
        }
        return best_dir;
//-------------------------------------------------
```

According to $a \leftarrow \text{argmax}\ \ \text{EVALUATE}(s, a')$,

Just test the four directions, then select the best action (make the value of r+(s′) maximize)

.

■ Describe how you implement AI::update_tuple_values()



```
            float error = 0.0;
//------------TO DO--------------------------------
            if (i == eb.size() - 1) {
                error = 0.0 - eb[i].sp.evaluate_score();
            }
            else {
                state st = eb[i].spp;
                int reward = st.move(get_best_move(st));
                error = reward + st.evaluate_score() - eb[i].sp.evaluate_score();
            }
//-------------------------------------------------
```

In terminal states, error = $0 - V(s^{'})$

Otherwise error = $R_{next} + V(s'_{next}) - V(s')$ and $s'_{next} = s''.move(a_{cur\_best\_action})$

- Statistic charts include following data
  - Winning rate and average score of standard tuple setting with 0.0025 learning rate (10%)

Average score: about 107000,

Max score: 17W~24W+

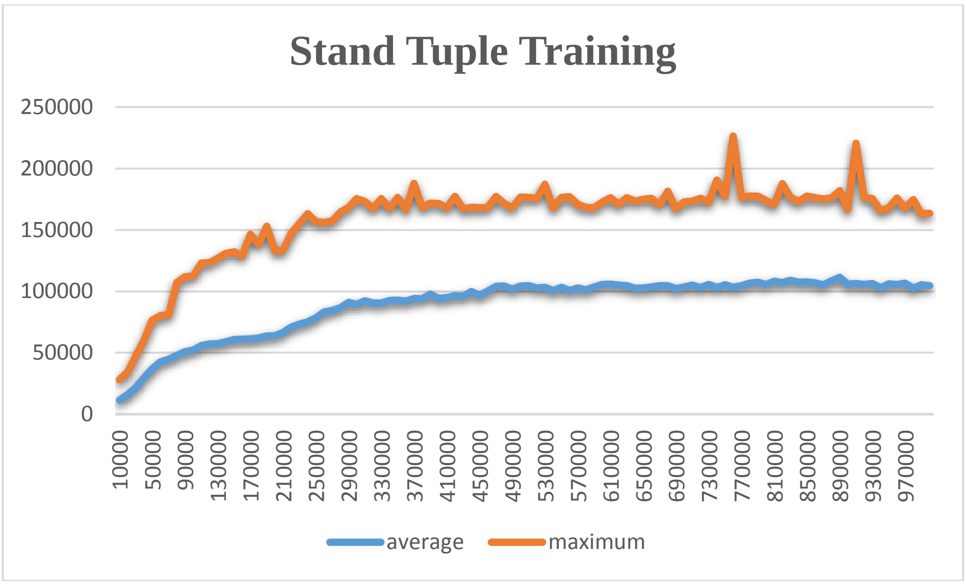


Figure 2: average & maximum scores

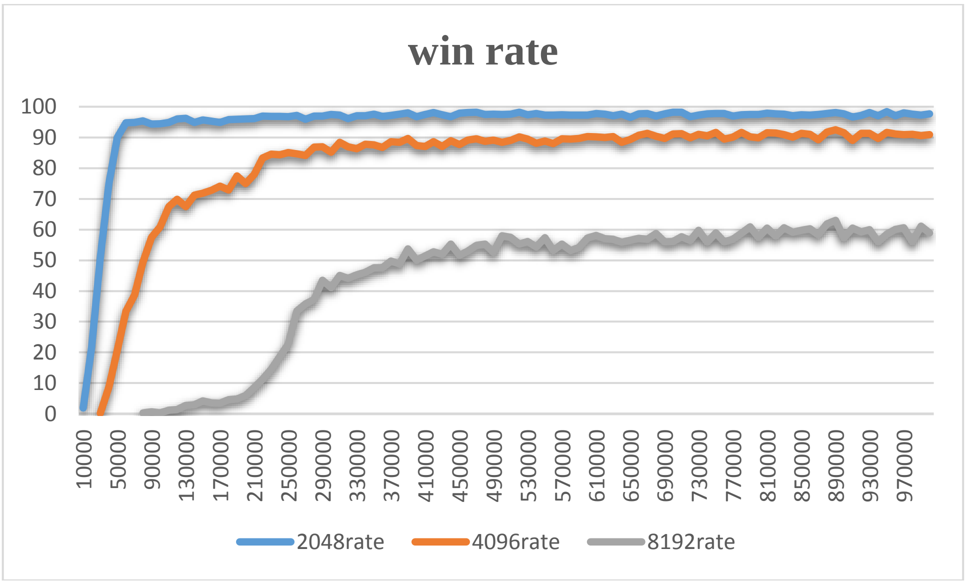
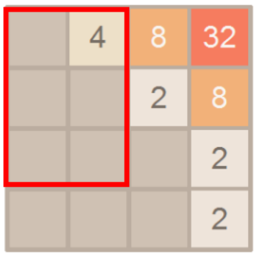
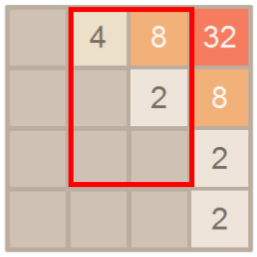


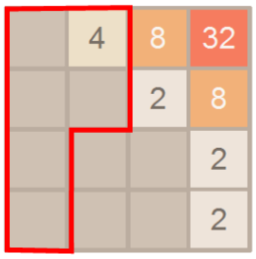
Figure 3: win rate of 2048 & 4096 & 8192

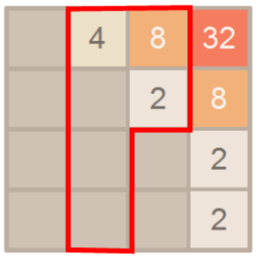
- Winning rate and average score of your tuple setting with learning rate 0.0025 (10%)

Tuples used:



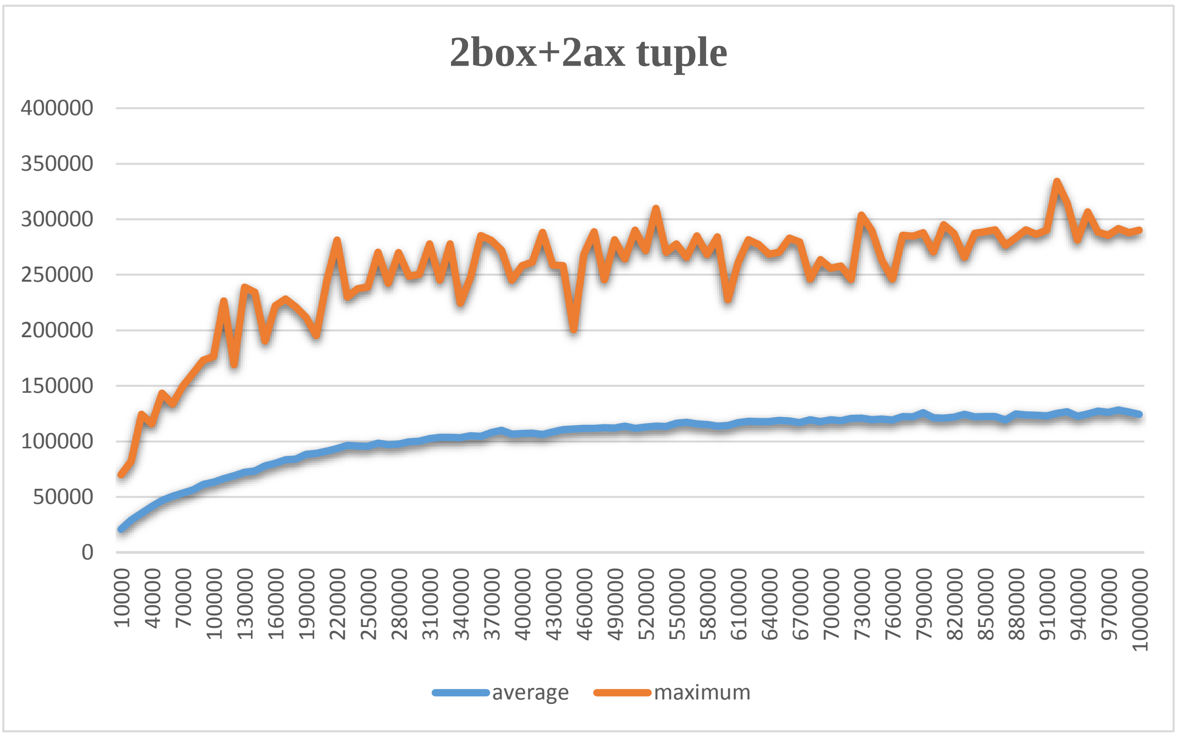


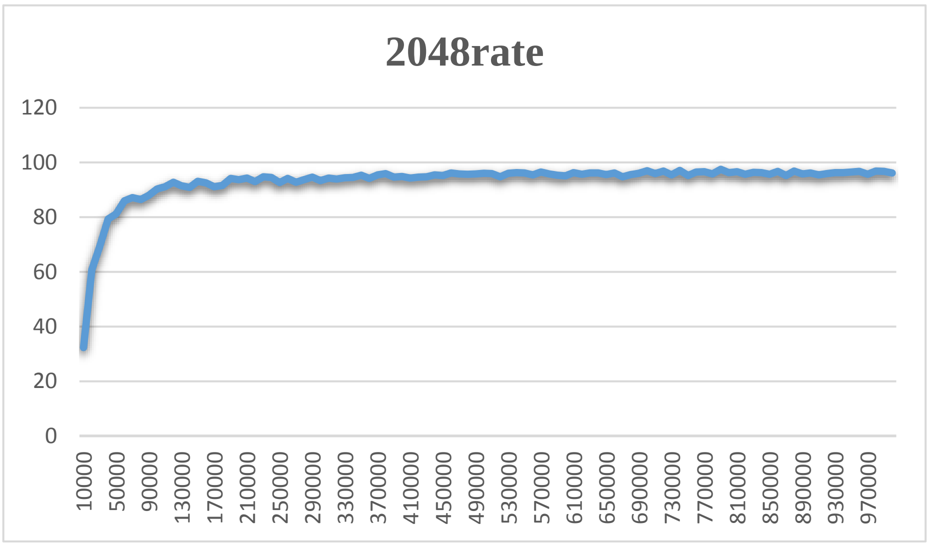




Average score: about 125000,

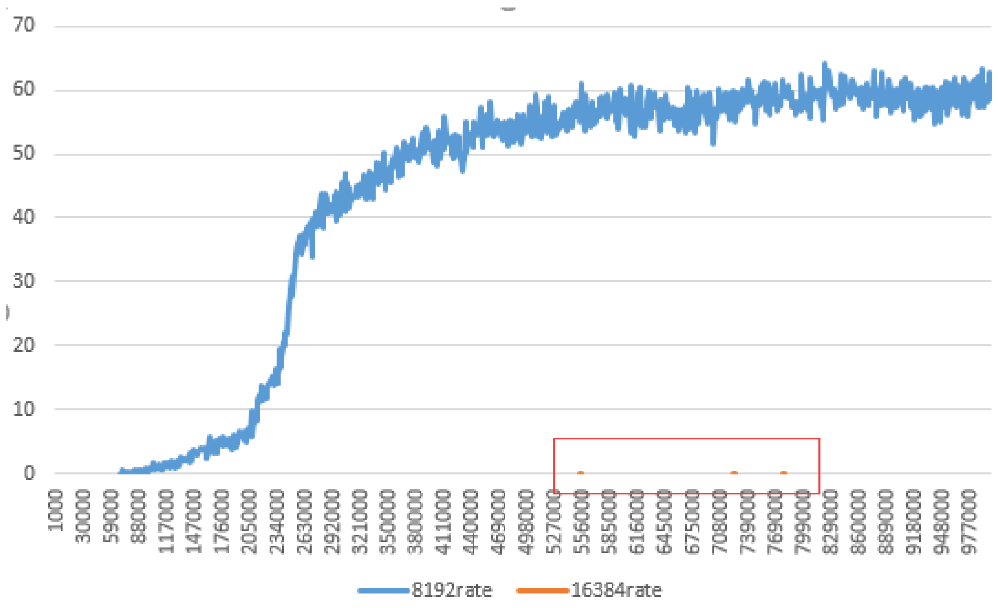
Max score: 25W~30W+



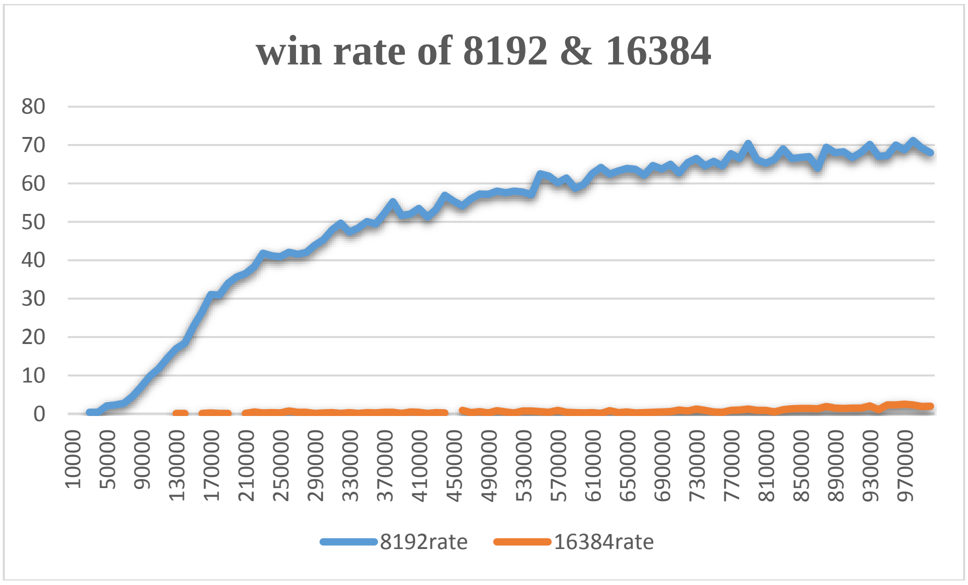


Discussion

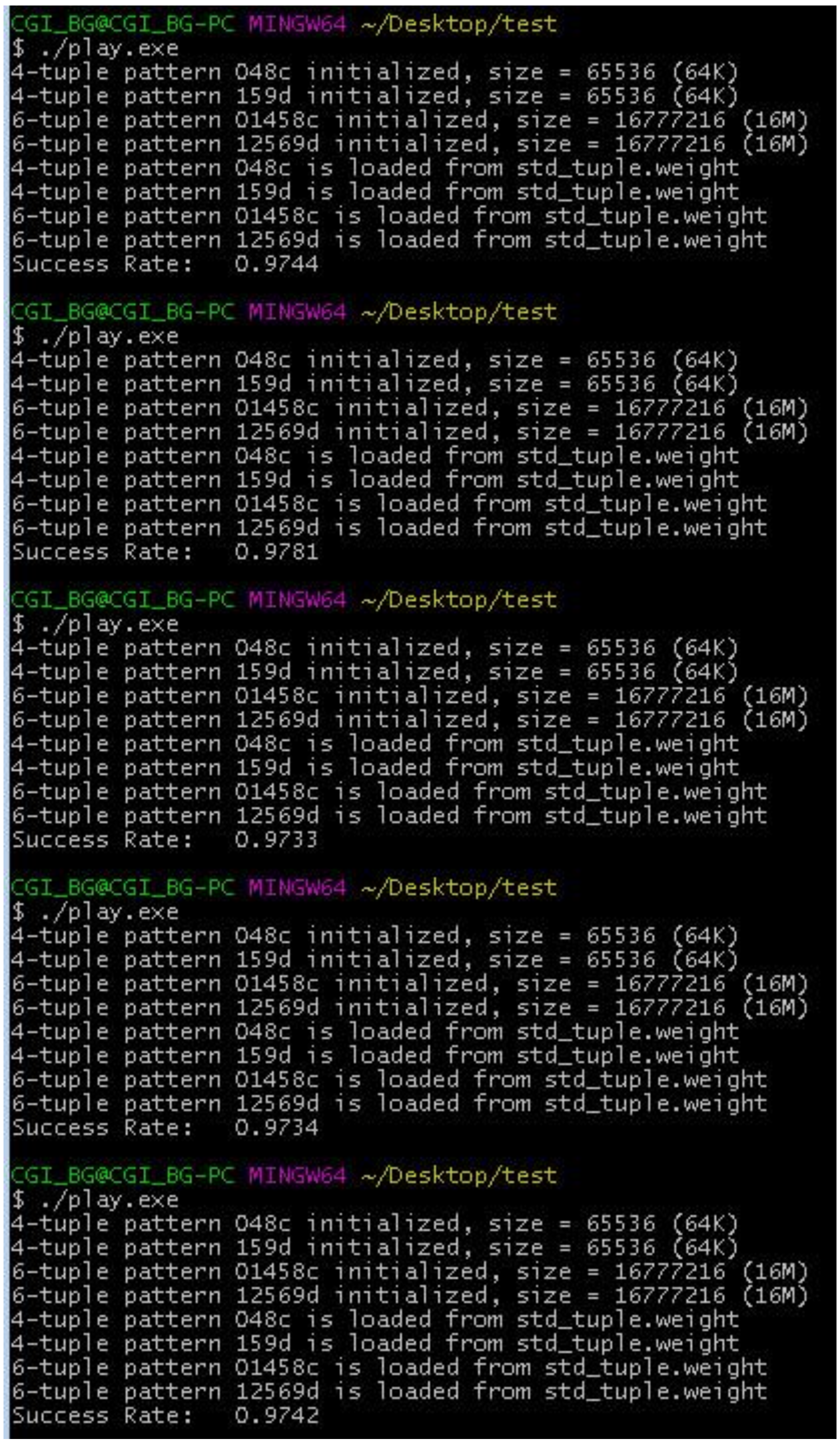
The STD tuples, we can't reach 16384 tile. However, I changed the $\gamma \rightarrow 0.98$, then i retrain the model for 100W iterations, sometimes the model reach 16384 tile.



About my tuple (2box + 2ax), it's performance is better than others.



Test



```
CGI_BG@CGI_BG-PC MINGW64 ~/Desktop/test
$ ./play.exe
4-tuple pattern 048c initialized, size = 65536 (64K)
4-tuple pattern 159d initialized, size = 65536 (64K)
6-tuple pattern 01458c initialized, size = 16777216 (16M)
6-tuple pattern 12569d initialized, size = 16777216 (16M)
4-tuple pattern 048c is loaded from std_tuple.weight
4-tuple pattern 159d is loaded from std_tuple.weight
6-tuple pattern 01458c is loaded from std_tuple.weight
6-tuple pattern 12569d is loaded from std_tuple.weight
Success Rate:   0.9744

CGI_BG@CGI_BG-PC MINGW64 ~/Desktop/test
$ ./play.exe
4-tuple pattern 048c initialized, size = 65536 (64K)
4-tuple pattern 159d initialized, size = 65536 (64K)
6-tuple pattern 01458c initialized, size = 16777216 (16M)
6-tuple pattern 12569d initialized, size = 16777216 (16M)
4-tuple pattern 048c is loaded from std_tuple.weight
4-tuple pattern 159d is loaded from std_tuple.weight
6-tuple pattern 01458c is loaded from std_tuple.weight
6-tuple pattern 12569d is loaded from std_tuple.weight
Success Rate:   0.9781

CGI_BG@CGI_BG-PC MINGW64 ~/Desktop/test
$ ./play.exe
4-tuple pattern 048c initialized, size = 65536 (64K)
4-tuple pattern 159d initialized, size = 65536 (64K)
6-tuple pattern 01458c initialized, size = 16777216 (16M)
6-tuple pattern 12569d initialized, size = 16777216 (16M)
4-tuple pattern 048c is loaded from std_tuple.weight
4-tuple pattern 159d is loaded from std_tuple.weight
6-tuple pattern 01458c is loaded from std_tuple.weight
6-tuple pattern 12569d is loaded from std_tuple.weight
Success Rate:   0.9733

CGI_BG@CGI_BG-PC MINGW64 ~/Desktop/test
$ ./play.exe
4-tuple pattern 048c initialized, size = 65536 (64K)
4-tuple pattern 159d initialized, size = 65536 (64K)
6-tuple pattern 01458c initialized, size = 16777216 (16M)
6-tuple pattern 12569d initialized, size = 16777216 (16M)
4-tuple pattern 048c is loaded from std_tuple.weight
4-tuple pattern 159d is loaded from std_tuple.weight
6-tuple pattern 01458c is loaded from std_tuple.weight
6-tuple pattern 12569d is loaded from std_tuple.weight
Success Rate:   0.9734

CGI_BG@CGI_BG-PC MINGW64 ~/Desktop/test
$ ./play.exe
4-tuple pattern 048c initialized, size = 65536 (64K)
4-tuple pattern 159d initialized, size = 65536 (64K)
6-tuple pattern 01458c initialized, size = 16777216 (16M)
6-tuple pattern 12569d initialized, size = 16777216 (16M)
4-tuple pattern 048c is loaded from std_tuple.weight
4-tuple pattern 159d is loaded from std_tuple.weight
6-tuple pattern 01458c is loaded from std_tuple.weight
6-tuple pattern 12569d is loaded from std_tuple.weight
Success Rate:   0.9742
```

std tuple test