
Final Project: Gomoku AI

106033233 周聖諺

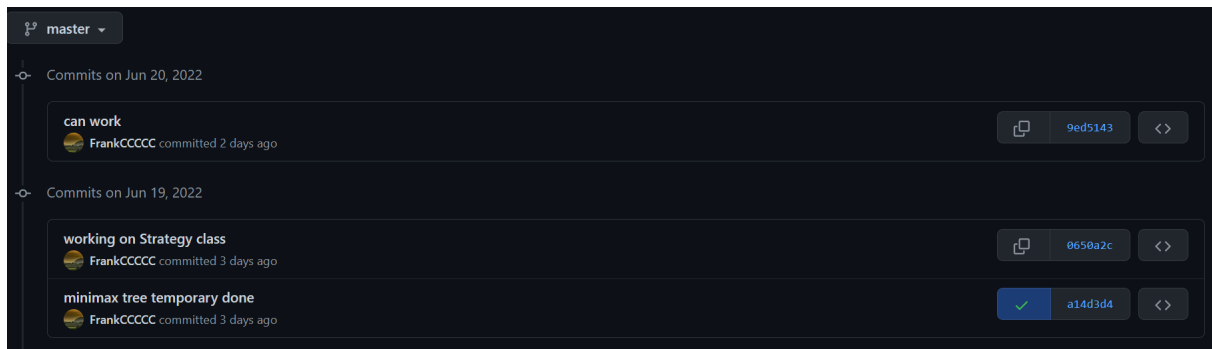
2022-06-22

Contents

Gomoku AI	3
Github	3
Threat Space Search	3
State Value Function	5
Minimax & Alpha-Beta Pruning (Negamax)	8
Negamax with alpha-beta pruning	9
Iterative Deepening	10
Zobrist Hash	10
Performance Issue	13
Reference	13

Gomoku AI

Github



Threat Space Search

Refers to Go-Moku and Threat-Space Search, 1994, L.V. Allis et.al

In some case, if you don't defense, then you will die unless you can get 5 in a row in 1 move.

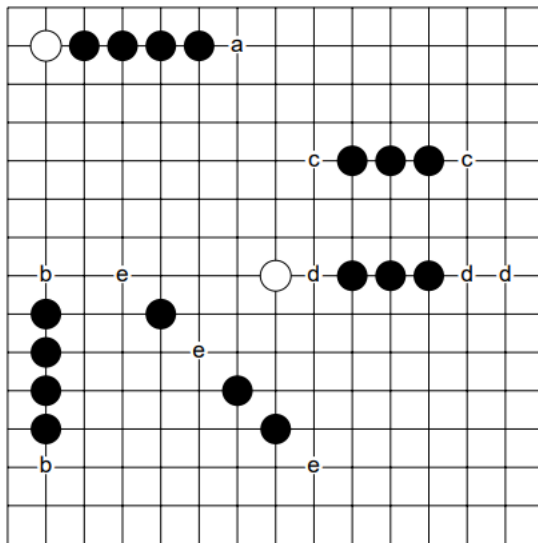


Diagram 1: Threats.

To win the game against any opposition a player needs to create a double threat (either a straight four, or two separate threats). In most cases, a threat sequence, i.e., a series of moves in which each consecutive move contains a threat, is played before a double threat occurs. A threat sequence leading to a (winning) double threat is called a *winning threat sequence*. Each threat in the sequence forces the defender to play a move countering the threat. Hence, the defender's possibilities are limited.

If you want to win, you need to pose double threats. In the ver1, I design a class `state` to record the board, state value and the candidates. But it's too slow to allocate a string.

Scan opponent's move to see whether the opponent poses a threat or not. If the opponent poses a threat, search that candidate move first(push the move into the head)

```

1  bool block_opponent = false;
2  int tmp_size = std::min(static_cast<int>(moves_opponent.size()), 2);
3  if (moves_opponent[0].score >= THRAT_SCORE_LIMIT) {
4      block_opponent = true;
5      for (int i = 0; i < tmp_size; ++i) {
6          auto move = moves_opponent[i];
7
8          // Re-evaluate move as current player
9          move.score = Eval::eval_pos(state, move.r, move.c, player);
10
11         // Add to candidate list
12         candidate_moves.push_back(move);
13     }

```

```
14 }
```

State Value Function

Since we know re-evaluate a state(whole board) is expensive and in evaluation, we actually compute the the same area, I design the state value function that only count the difference of the board, which is the move of the AI and the opponent.

Each move will affect a star area nearby. So, I measure it in 4 directions, which are horizontal, vertical, diagonal directions.

```
1 void Eval::gen_measures(const char *state, int r, int c, int player,
2   bool is_cont, Eval::Measure *ms) {
3   ERR_NULL_CHECK(state,)
4   ERR_POS_CHECK(r,c,)
5   // Scan 4 directions
6   gen_measure(state, r, c, Eval::MEASURE_DIR_H, player, is_cont, ms
7     [0]);
8   gen_measure(state, r, c, Eval::MEASURE_DIR_LU, player, is_cont, ms
9     [1]);
10  gen_measure(state, r, c, Eval::MEASURE_DIR_V, player, is_cont, ms
11    [2]);
12  gen_measure(state, r, c, Eval::MEASURE_DIR_RU, player, is_cont, ms
13    [3]);
14 }
```

Each time I record {Number of pieces in a row, Number of ends blocked by edge or the other player (0-2), Number of spaces in the middle of pattern}

```
1 // Result of a measurement in a direction
2 struct Measure {
3   // Number of pieces in a row
4   char len;
5   // Number of ends blocked by the other player (1 or 2) or the
6   // border of the board (-1: don't care the value)
7   char block_cnt;
8   // Number of empty spaces inside the row(number of pieces to
9   // separate the row, -1: don't care the value)
10  char space_cnt;
11 };
```

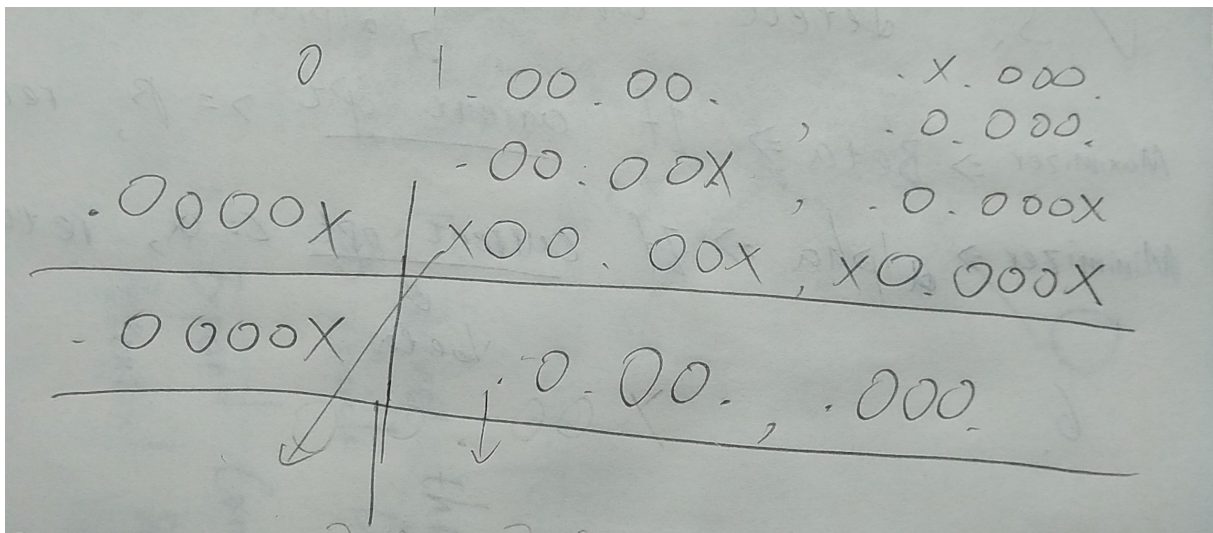
And I define the pattern as {Length of pattern (pieces in a row), Number of ends blocked by edge or the other player (0-2), Number of spaces in the middle of pattern (-1: Ignore value)}

```
1 // A pattern in a direction
```

```

2 struct Pattern {
3     // Minimum count of occurrences
4     char min_occur;
5     // Length of a pattern (number of pieces in a row)
6     char len;
7     // Number of ends blocked by the other player (1 or 2) or the
        border of the board (-1: don't care the value)
8     char block_cnt;
9     // Number of empty spaces inside the row(number of pieces to
        separate the row, -1: don't care the value)
10    char space_cnt;
11 };
12 const Eval::Pattern *Eval::PATTERNS = new Eval::Pattern[PATTERNS_NUM *
        2]{
13     {1, 5, 0, 0}, {0, 0, 0, 0}, // 10000
14     {1, 4, 0, 0}, {0, 0, 0, 0}, // 700
15     // Threats-----
16     {2, 4, 1, 0}, {0, 0, 0, 0}, // 700
17     {2, 4, -1, 1}, {0, 0, 0, 0}, // 700
18     {1, 4, 1, 0}, {1, 4, -1, 1}, // 700
19     {1, 4, 1, 0}, {1, 3, 0, -1}, // 500
20     {1, 4, -1, 1}, {1, 3, 0, -1}, // 500
21     {2, 3, 0, -1}, {0, 0, 0, 0}, // 300
22     // Threats-----
23     ...

```



Evaluate the given position

```

1 int Eval::eval_pos(const char *state, int r, int c, int player) {
2     // Check parameters
3     ERR_NULL_CHECK(state, 0)
4     ERR_PLAYER_CHECK(player, 0)
5
6     Measure ms[M_DIR_NUM];

```

```

7
8    // Measure continuous and non-continuous conditions
9    gen_measures(state, r, c, player, false, ms);
10   Score sc_non_conti = eval_measures(ms);
11   gen_measures(state, r, c, player, true, ms);
12   Score sc_conti = eval_measures(ms);
13   return std::max(sc_non_conti, sc_conti);
14 }
15
16 int Eval::eval_measures(const Measure *measure_4d) {
17     int sc = 0;
18
19     // Find the longest length in measure_4d and skip some patterns
20     // that is longer than the Measure
21     int max_measure_len = 0;
22     for (int i = 0; i < M_DIR_NUM; i++) {
23         int len = measure_4d[i].len;
24         max_measure_len = max(len, max_measure_len);
25         sc += (len - 1);
26     }
27     int start_pat_idx = SKIP_PATTERNS[max_measure_len];
28
29     // Match specified patterns, ignore the patterns measures doesn't
30     // have
31     for (int i = start_pat_idx; i < PATTERNS_NUM; i++) {
32         sc += match_pattern(measure_4d, &PATTERNS[2 * i]) *
33             PATTERN_SCORES[i];
34     }
35
36     // Only match one threatening pattern
37     if (sc >= THRAT_SCORE_LIMIT){break;}
```

How I count the space and the block of a sequence

```

1 for (int i = 0; i < 2; i++) {
2     while (true) {
3         // Shift
4         r_cnt += dr; c_cnt += dc;
5
6         // Validate position
7         if (pos_check(r_cnt, c_cnt)){break;}
```

```

        c_cnt + dc) == player) {
15         allowed_space--;
16         res.space_cnt++;
17         continue;
18     } else {
19         res.block_cnt--;
20         break;
21     }
22 }
23
24 // Another player
25 if (spot != player){break;}
26
27 // Current player
28 res.len++;
29 }
30
31 // Reverse direction
32 dr = -dr;
33 dc = -dc;
34 r_cnt = r;
35 c_cnt = c;
36 }

```

Accumulate the scores along trajectory. Because the `Negamax::negamax` will return opponent' s score(score of opponent' s utility), we need to minus it.

```

1  if (depth > 1) sc = negamax(state, opn, initial_depth, depth - 1,
    enable_ab_pruning, -beta, -alpha, dummy_r, dummy_c);
2
3  // Decay longer moves
4  sc = static_cast<int>(sc * SCORE_DECAY);
5
6  // Calculate score difference
7  move.accum_score = move.score - sc;
8
9  // Store back to candidate array
10 cand_mvs.at(i).accum_score = move.accum_score;

```

Minimax & Alpha-Beta Pruning (Negamax)

It' s just another implementation of Minimax. It' s based on the following formula

$$\max(a, b) = -\min(-a, -b)$$

Negamax Search

- $\min\{a_0, \dots, a_n\} = -\max\{-a_0, \dots, -a_n\}$
- Such simplified implementation of MINIMAX is called NEGAMAX.
- Copying the whole state (line 5) is memory consuming. Practical implementation usually adopts $s = \text{BACKTRACK}(s', a)$.

NEGAMAX(s)

```

1  if TERMINAL-TEST( $s$ )
2    return UTILITY( $s, p$ )
3  result =  $-\infty$ 
4  for each  $a \in \text{ACTION}(s)$ 
→ 5     $s' = \text{RESULT}(s, a)$ 
6    result = max(result, -NEGAMAX( $s'$ ))
7  return result

```

Pseudo code

```

1  function negamax(node, depth, color) is
2    if depth = 0 or node is a terminal node then
3      return color × the heuristic value of node
4    value :=  $-\infty$ 
5    for each child of node do
6      value := max(value, -negamax(child, depth - 1, -color))
7    return value

```

Negamax with alpha-beta pruning

Pseudo code

```

1  function negamax(node, depth,  $\alpha$ ,  $\beta$ , color) is
2    if depth = 0 or node is a terminal node then
3      return color × the heuristic value of node
4
5    childNodes := generateMoves(node)
6    childNodes := orderMoves(childNodes)
7    value :=  $-\infty$ 
8    foreach child in childNodes do
9      value := max(value, -negamax(child, depth - 1, - $\beta$ , - $\alpha$ , -color))
10      $\alpha$  := max( $\alpha$ , value)
11     if  $\alpha \geq \beta$  then
12       break (* cut-off *)
13    return value

```

```

1  sc = negamax(state,

```

```

2         opponent,          // Change player
3         initial_depth,      // Initial depth
4         depth - 1,          // Reduce depth by 1
5         enable_ab_pruning,  // Alpha-Beta
6         -beta,              //
7         -alpha,             //
8         move_r,             // Result move
9         move_c);
10
11 // Store back to candidate array
12 cand_mvs.at(i).accum_score = move.accum_score;
13
14 // Restore the move
15 Util::set_spot(state, move.r, move.c, 0);
16
17 // Update maximum score
18 if (move.accum_score > max_score) {
19     max_score = move.accum_score;
20     move_r = move.r; move_c = move.c;
21 }
22
23 // Alpha-beta
24 if (max_score > alpha) alpha = max_score;
25 if (enable_ab_pruning && alpha >= beta) break;

```

Iterative Deepening

Re-search the game tree deeper when the time is enough

```

1 for (int d = INIT_DEPTH;; d += INC_DEPTH) {
2     // Reset game state
3     memcpy(ng_state, state, G_B_AREA);
4
5     // Execute search
6     negamax(ng_state, player, d, d, enable_ab_pruning, alpha, beta,
7             move_r, move_c);
8     actual_depth = d;
9     INFO("Deepening - actual_depth: " << actual_depth << " Act: (" <<
10         move_r << ", " << move_c << ") " << " node_count: " << g_node_cnt
11         << " eval_count: " << g_eval_cnt)
12     io.write_valid_spot(Position(move_r, move_c));
13 }

```

Zobrist Hash

I've tried to cache the searched state in a hash table with Zobrist hashing as the key. However, the `operator[]` and `find()` takes too much time to insert and check whether the key is in the table or

not. Although it reduce the time of the `eval_pos()` but it takes much more time to handling the hash table. The first figure is the profiling of the hash table caching and the second one is the one without hash table.

CPU Usage

Current View: Call Tree Expand Hot Path Show Hot Path Reset Root Search

Function Name	Total CPU [unit, %]	Self CPU [unit, %]	Module	Category
negamax:negamax	31307 (13.14%)	0 (0.01%)	project3_new.exe	File System Kernel
Negamax:negamax	31067 (66.41%)	5 (0.01%)	project3_new.exe	File System Kernel
Negamax:negamax	19496 (41.67%)	10 (0.02%)	project3_new.exe	File System Kernel
Negamax:search_sorted_cands	19353 (41.37%)	92 (0.20%)	project3_new.exe	File System Kernel
Eval:eval_pos	15981 (34.16%)	182 (0.39%)	project3_new.exe	File System Kernel
std::unordered_map<unsigned ...	6991 (14.94%)	7 (0.01%)	project3_new.exe	File System Kernel
std::Hash<std::Umap_traits<u...	4726 (10.10%)	29 (0.06%)	project3_new.exe	Kernel
Eval:eval_pos	2385 (5.10%)	40 (0.09%)	project3_new.exe	Kernel
std::List_iterator<std::List_val<...	810 (1.73%)	16 (0.03%)	project3_new.exe	Kernel
std::Hash<std::Umap_traits<u...	547 (1.17%)	20 (0.04%)	project3_new.exe	Kernel
std::List_iterator<std::List_val<...	90 (0.19%)	3 (0.01%)	project3_new.exe	Kernel
std::List_const_iterator<std::Lis...	68 (0.15%)	22 (0.05%)	project3_new.exe	Kernel
std::List_iterator<std::List_val<...	65 (0.14%)	64 (0.14%)	project3_new.exe	Kernel
ZobristHash::yield	51 (0.11%)	26 (0.06%)	project3_new.exe	Kernel
_RTC_CheckStackVars	28 (0.06%)	27 (0.06%)	project3_new.exe	Kernel
_CheckForDebuggerJustMyCode	17 (0.04%)	17 (0.04%)	project3_new.exe	Kernel
_security_check_cookie	8 (0.02%)	8 (0.02%)	project3_new.exe	Kernel
ILT.ILT+2110(?_autoclassinit2	3 (0.01%)	3 (0.01%)	project3_new.exe	Kernel
ILT.ILT+720(?71?\$_List_iterator	3 (0.01%)	3 (0.01%)	project3_new.exe	Kernel
ILT.ILT+2390(?7C?\$_List_iterator	2 (0.00%)	2 (0.00%)	project3_new.exe	Kernel
[System Call] ntoskrnl.exe	1 (0.00%)	1 (0.00%)	ntoskrnl.exe	Kernel
ILT.ILT+2090(_security_check_c...	1 (0.00%)	1 (0.00%)	project3_new.exe	Kernel
ILT.ILT+3475(_RTC_CheckStackVa...	1 (0.00%)	1 (0.00%)	project3_new.exe	Kernel
ILT.ILT+4165(?\$_find	1 (0.00%)	1 (0.00%)	project3_new.exe	Kernel
ILT.ILT+4195(_CheckForDebugg...	1 (0.00%)	1 (0.00%)	project3_new.exe	Kernel
std::sort<std::Vector_iterator<std...	1550 (3.31%)	2 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::al...	1306 (2.79%)	11 (0.02%)	project3_new.exe	File System Kernel
Util::remote_spot	339 (0.72%)	326 (0.70%)	project3_new.exe	Kernel

CPU Usage

Current View: Call Tree Expand Hot Path Show Hot Path Reset Root Search

Function Name	Total CPU [unit, %]	Self CPU [unit, %]	Module	Category
Negamax:negamax	21759 (97.56%)	0 (0.00%)	project3_new.exe	Kernel
Negamax:negamax	21739 (97.47%)	0 (0.00%)	project3_new.exe	Kernel
Negamax:negamax	21712 (97.35%)	0 (0.00%)	project3_new.exe	Kernel
Negamax:negamax	21647 (97.06%)	1 (0.00%)	project3_new.exe	Kernel
Negamax:negamax	21247 (95.27%)	1 (0.00%)	project3_new.exe	Kernel
Negamax:negamax	20092 (90.09%)	1 (0.00%)	project3_new.exe	Kernel
Negamax:negamax	17816 (79.88%)	8 (0.04%)	project3_new.exe	Kernel
Negamax:negamax	13188 (59.13%)	7 (0.03%)	project3_new.exe	Kernel
Negamax:negamax	8248 (36.98%)	5 (0.02%)	project3_new.exe	Kernel
Negamax:search_sorted_cands	8136 (36.48%)	88 (0.39%)	project3_new.exe	Kernel
Eval:eval_pos	4938 (22.14%)	89 (0.40%)	project3_new.exe	Kernel
std::sort<std::Vector_iterator<std...	1548 (6.94%)	0 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::al...	1154 (5.17%)	14 (0.06%)	project3_new.exe	Kernel
Util::remote_spot	337 (1.51%)	327 (1.47%)	project3_new.exe	Kernel
active_area	38 (0.17%)	37 (0.17%)	project3_new.exe	Kernel
ZobristHash::ZobristHash	28 (0.13%)	0 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::al...	3 (0.01%)	0 (0.00%)	project3_new.exe	Kernel
_RTC_CheckStackVars	1 (0.00%)	1 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::al...	1 (0.00%)	0 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::allo...	35 (0.16%)	0 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::allo...	29 (0.13%)	0 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::allo...	23 (0.10%)	0 (0.00%)	project3_new.exe	Kernel
Eval:eval_pos	14 (0.06%)	0 (0.00%)	project3_new.exe	Kernel
ZobristHash::ZobristHash	3 (0.01%)	0 (0.00%)	project3_new.exe	Kernel
std::vector<Negamax::Move,std::allo...	1 (0.00%)	1 (0.00%)	project3_new.exe	Kernel

D:\data\2P_kuo\project3\project3_new\project3_new\main.cpp:792

```

785 // if (actual_depth != nullptr) *actual_depth = d;
792 actual_depth = d;

```

85 % No issues found

I've also measure the hit rate of the hash table, it shows that about 60% of queries are missed in the shallower depth but only 40% of queries are missed in the deeper depth. As a result, the hash table

may yield benefit in the deeper game tree. But consider the constraint of the project, we only have 10s. The hash table cannot help.

```
Microsoft Visual Studio Debug Console
Deepening - actual_depth: 6 | Act: (10, 6) | node_count: 1154 | eval_count: 361350 | eval_missed: 217428(60.171%) | Iter ms: 1611
Deepening - actual_depth: 8 | Act: (10, 6) | node_count: 9703 | eval_count: 3024658 | eval_missed: 1475084(48.7686%) | Iter ms: 6252
Deepening - actual_depth: 10 | Act: (10, 6) | node_count: 53074 | eval_count: 16466934 | eval_missed: 6934720(42.113%) | Iter ms: 28852
Success actual_depth: 10 Act: (10, 6) winning_player: 0 node_count: 53074 eval_count: 16466934

D:\data\I2P_Kuo\project3\project3_new\x64\Debug\project3_new.exe (process 15980) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

```
Microsoft Visual Studio Debug Console
Deepening - actual_depth: 6 | Act: (10, 6) | node_count: 1154 | eval_count: 361350 | eval_missed: 0(0%) | Iter ms: 914
Deepening - actual_depth: 8 | Act: (10, 6) | node_count: 9703 | eval_count: 3024658 | eval_missed: 0(0%) | Iter ms: 4174
Deepening - actual_depth: 10 | Act: (10, 6) | node_count: 53074 | eval_count: 16466934 | eval_missed: 0(0%) | Iter ms: 17634
Success actual_depth: 10 Act: (10, 6) winning_player: 0 node_count: 53074 eval_count: 16466934

D:\data\I2P_Kuo\project3\project3_new\x64\Debug\project3_new.exe (process 13512) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

```
1 ZobristHash::ZobristHash(const char *state) {
2     this->state_hash = ZobristHash::zobrist_hash(state);
3 }
4
5 ZobristHash::ClassInit::ClassInit() {
6     // Static constructor definition
7     std::random_device rd;
8     std::mt19937 gen(rd());
9     std::uniform_int_distribution<ZbsHash> d(0, UINT64_MAX);
10
11     ZobristHash::HASH_O = new ZbsHash[G_B_AREA];
12     ZobristHash::HASH_X = new ZbsHash[G_B_AREA];
13     ZobristHash::HASH_R = new ZbsHash[G_B_SIZE];
14     ZobristHash::HASH_C = new ZbsHash[G_B_SIZE];
15     ZobristHash::HASH_ROLE = new ZbsHash[ROLE_NUM];
16
17     // Generate random values
18     for (int i = 0; i < G_B_AREA; i++) {
19         ZobristHash::HASH_O[i] = d(gen);
20         ZobristHash::HASH_X[i] = d(gen);
21     }
22     for (int i = 0; i < G_B_SIZE; i++) {
23         ZobristHash::HASH_R[i] = d(gen);
24         ZobristHash::HASH_C[i] = d(gen);
25     }
26     for (int i = 0; i < ROLE_NUM; i++){
27         ZobristHash::HASH_ROLE[i] = d(gen);
28     }
29 }
30
31 ZbsHash ZobristHash::zobrist_hash(const char *state) {
```

```

32     ZbsHash h = 0;
33     for (int i = 0; i < G_B_AREA; i++) {
34         if (state[i] == 1) { h ^= HASH_O[i]; }
35         else if (state[i] == 2) { h ^= HASH_X[i]; }
36     }
37     return h;
38 }
39 Hash ZobristHash::hash(const char *state, int r, int c, int player) {
40     Hash h = ZobristHash::zobrist_hash(state);
41     h ^= (ZobristHash::HASH_R[r] ^ ZobristHash::HASH_C[c] ^ ZobristHash::HASH_ROLE[player]);
42     return h;
43 }
44 Hash ZobristHash::yield(int r, int c, int player) const {
45     return this->state_hash ^ (ZobristHash::HASH_R[r] ^ ZobristHash::HASH_C[c] ^ ZobristHash::HASH_ROLE[player]);
46 }
47 std::string hash_str(const char *state, int r, int c, int player) {
48     char rc = (r + '0');
49     char cc = (c + '0');
50     char pc = (player + '0');
51     return std::string(state) + rc + cc + pc;
52 }
53 // ZbsHash *ZobristHash::HASH_O, *ZobristHash::HASH_X;
54 ZbsHash *ZobristHash::HASH_O, *ZobristHash::HASH_X, *ZobristHash::HASH_R, *ZobristHash::HASH_C, *ZobristHash::HASH_ROLE;
55 ZobristHash::ClassInit ZobristHash::Initialize;

```

Performance Issue

- Allocate `std::string`/ append string are very expensive(half of runtime in ver1)
- Allocate class instance is much more expensive
- Use structure, inline function, static method and static variables as much as possible

Reference

- Pandoc compile command

```

1 pandoc -o readme.pdf readme.md --pdf-engine=xelatex -V CJKmainfont="Microsoft JhengHei" --from markdown --template eisvogel --listings --toc --toc-depth=4

```