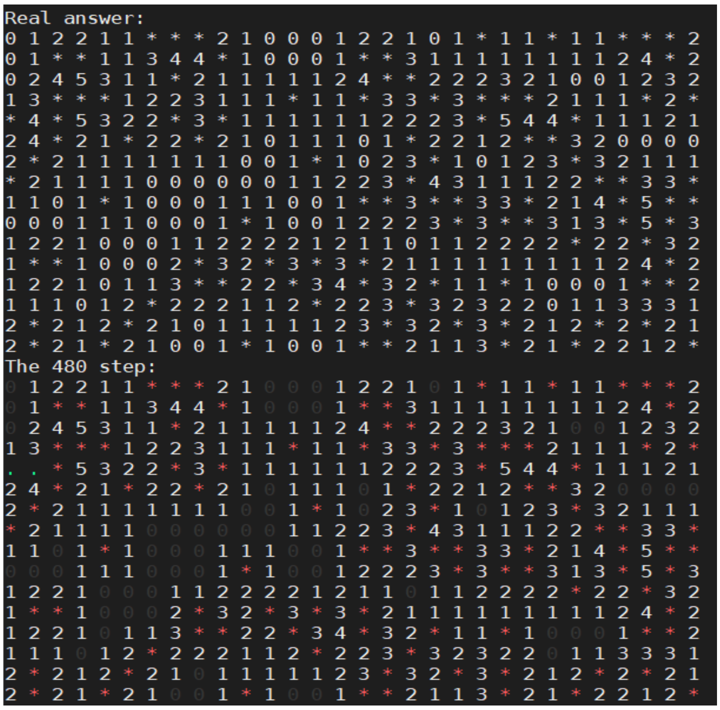
Programming Assignment #3

0612129 郭家佑

Demo:

Just like the following image, the executable file usage is ./minesweeper <difficulty> and the difficulty argument is same as spec, 1 (Easy, 9x9 board with 10 mines),2 (Medium, 16x16 board with 25 mines), and 3(Hard, 30x16 board with 99 mines), otherwise I add a 4th level, 4 (self-defined, after run the program, user can input x size, y size, and mines number). And it would dump two table to screen, first one is the initial table, and the second is the end game table. If you want to see all the process just only comment the if condition (main.cpp, line 22).

I used the "\x1B[??m" to color the mines (red) and not sure position (green) and also darken the hints which is zero just like the game we used to play (minesweeper.cpp, line 416) .



```
Real answer:
0 1 2 2 1 1 * * * 2 1 0 0 0 1 2 2 1 0 1 * 1 1 * 1 1 * * * 2
0 1 * * 1 1 3 4 4 * 1 0 0 0 1 * * 3 1 1 1 1 1 1 1 1 2 4 * 2
0 2 4 5 3 1 1 * 2 1 1 1 1 1 2 4 * * 2 2 2 3 2 1 0 0 1 2 3 2
1 3 * * * 1 2 2 3 1 1 1 * 1 1 * 3 3 * 3 * * * 2 1 1 1 * 2 *
* 4 * 5 3 2 2 * 3 * 1 1 1 1 1 1 2 2 2 3 * 5 4 4 * 1 1 1 2 1
2 4 * 2 1 * 2 2 * 2 1 0 1 1 1 0 1 * 2 2 1 2 * * 3 2 0 0 0 0
2 * 2 1 1 1 1 1 1 1 0 0 1 * 1 0 2 3 * 1 0 1 2 3 * 3 2 1 1 1
* 2 1 1 1 1 0 0 0 0 0 0 1 1 2 2 3 * 4 3 1 1 1 2 2 * * 3 3 *
1 1 0 1 * 1 0 0 0 1 1 1 0 0 1 * * 3 * * 3 3 * 2 1 4 * 5 * *
0 0 0 1 1 1 0 0 0 1 * 1 0 0 1 2 2 2 3 * 3 * * 3 1 3 * 5 * 3
1 2 2 1 0 0 0 1 1 2 2 2 2 1 2 1 1 0 1 1 2 2 2 2 * 2 2 * 3 2
1 * * 1 0 0 0 2 * 3 2 * 3 * 3 * 2 1 1 1 1 1 1 1 1 1 2 4 * 2
1 2 2 1 0 1 1 3 * * 2 2 * 3 4 * 3 2 * 1 1 * 1 0 0 0 1 * * 2
1 1 1 0 1 2 * 2 2 2 1 1 2 * 2 2 3 * 3 2 3 2 2 0 1 1 3 3 3 1
2 * 2 1 2 * 2 1 0 1 1 1 1 1 2 3 * 3 2 * 3 * 2 1 2 * 2 * 2 1
2 * 2 1 * 2 1 0 0 1 * 1 0 0 1 * * 2 1 1 3 * 2 1 * 2 2 1 2 *
The 480 step:
0 1 2 2 1 1 * * * 2 1 0 0 0 1 2 2 1 0 1 * 1 1 * 1 1 * * * 2
0 1 * * 1 1 3 4 4 * 1 0 0 0 1 * * 3 1 1 1 1 1 1 1 1 2 4 * 2
0 2 4 5 3 1 1 * 2 1 1 1 1 1 2 4 * * 2 2 2 3 2 1 0 0 1 2 3 2
1 3 * * * 1 2 2 3 1 1 1 * 1 1 * 3 3 * 3 * * * 2 1 1 1 * 2 *
. . * 5 3 2 2 * 3 * 1 1 1 1 1 1 2 2 2 3 * 5 4 4 * 1 1 1 2 1
2 4 * 2 1 * 2 2 * 2 1 0 1 1 1 0 1 * 2 2 1 2 * * 3 2 0 0 0 0
2 * 2 1 1 1 1 1 1 1 0 0 1 * 1 0 2 3 * 1 0 1 2 3 * 3 2 1 1 1
* 2 1 1 1 1 0 0 0 0 0 0 1 1 2 2 3 * 4 3 1 1 1 2 2 * * 3 3 *
1 1 0 1 * 1 0 0 0 1 1 1 0 0 1 * * 3 * * 3 3 * 2 1 4 * 5 * *
0 0 0 1 1 1 0 0 0 1 * 1 0 0 1 2 2 2 3 * 3 * * 3 1 3 * 5 * 3
1 2 2 1 0 0 0 1 1 2 2 2 2 1 2 1 1 0 1 1 2 2 2 2 * 2 2 * 3 2
1 * * 1 0 0 0 2 * 3 2 * 3 * 3 * 2 1 1 1 1 1 1 1 1 1 2 4 * 2
1 2 2 1 0 1 1 3 * * 2 2 * 3 4 * 3 2 * 1 1 * 1 0 0 0 1 * * 2
1 1 1 0 1 2 * 2 2 2 1 1 2 * 2 2 3 * 3 2 3 2 2 0 1 1 3 3 3 1
2 * 2 1 2 * 2 1 0 1 1 1 1 1 2 3 * 3 2 * 3 * 2 1 2 * 2 * 2 1
2 * 2 1 * 2 1 0 0 1 * 1 0 0 1 * * 2 1 1 3 * 2 1 * 2 2 1 2 *
```

If you want to re-compile it, only need to input “make” and used the makedile

Observations and interpretations:

Thought professor say the stuck situation is very rare, but in my test almost half of test would stuck, of course, the situation only happened on difficulty 3 in the other 2 level almost no any fault case. And the following table is showing after I run the test.sh and get data from data.txt which information are written by main.cpp statement (line 29).

The table show that although halt of tests is not reach the end step, but they still complete most of game. And the average time it spends is about 2 min.

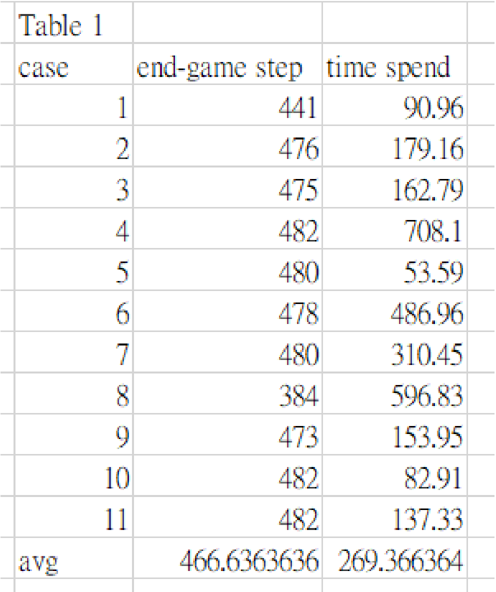


Table 1		
case	end-game step	time spend
1	441	90.96
2	476	179.16
3	475	162.79
4	482	708.1
5	480	53.59
6	478	486.96
7	480	310.45
8	384	596.83
9	473	153.95
10	482	82.91
11	482	137.33
avg	466.6363636	269.366364

Although 2 min is acceptable for me, but I still want to speed up the program. Then, the part about matching call to my mind. In the spec about matching part, one clause with only at most two literals and the other one have no limited. Then, if I choose both two clauses only have most two literals?

Then, only one game isn't stuck (max step is 482, 16*30 + 1 for original state and 1 for end game state which used for check there are no addition position can choose), and the average game step reduced to 417 step. And about 80 percent game can arrive more than 400 step, it may show that the it can't solve some complex clause in the situation.

However, the time spending only about 8 second, reducing extremely.

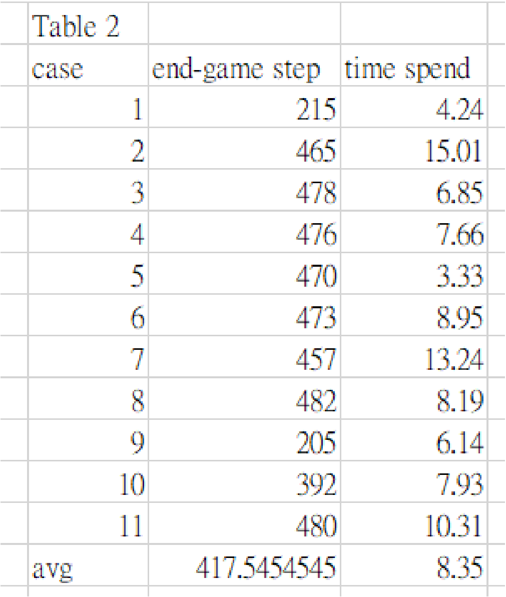


Table 2		
case	end-game step	time spend
1	215	4.24
2	465	15.01
3	478	6.85
4	476	7.66
5	470	3.33
6	473	8.95
7	457	13.24
8	482	8.19
9	205	6.14
10	392	7.93
11	480	10.31
avg	417.5454545	8.35

Last, I want to try the case without any matching
In the following table (3), only one game arrives to more than 400 step.
And average step also less than 300. Although one game only spends not more than 0.5 second, but it also useless.

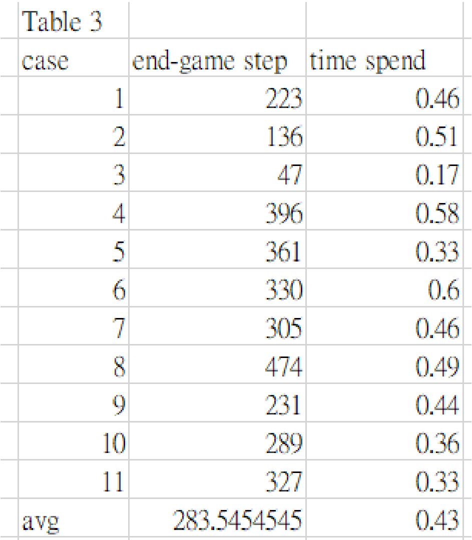
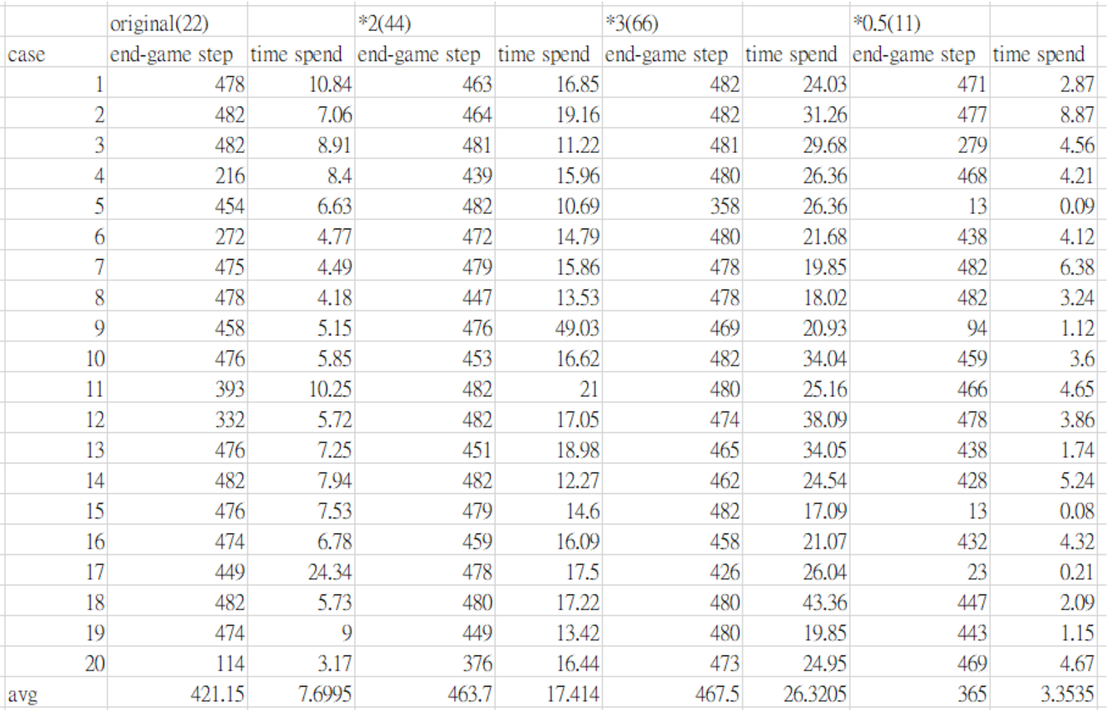


Table 3		
case	end-game step	time spend
1	223	0.46
2	136	0.51
3	47	0.17
4	396	0.58
5	361	0.33
6	330	0.6
7	305	0.46
8	474	0.49
9	231	0.44
10	289	0.36
11	327	0.33
avg	283.5454545	0.43

Of course, those data of test case are random due to the srand(time(0)); in the minesweeper.cpp (line 11).

Otherwise, I also tested the case we change about initial safe cells, in original case, round(sqrt(#cells)) can make the easy and medium level trivial. But it is not used for bigger size, like 30*30, it often stuck.
In the following table, there are four test case about double, triple and half of safe cells. Those test all use the way of table 2 to reduced time to run and also make the data have more obvious different.



	original(22)		*2(44)		*3(66)		*0.5(11)	
case	end-game step	time spend	end-game step	time spend	end-game step	time spend	end-game step	time spend
1	478	10.84	463	16.85	482	24.03	471	2.87
2	482	7.06	464	19.16	482	31.26	477	8.87
3	482	8.91	481	11.22	481	29.68	279	4.56
4	216	8.4	439	15.96	480	26.36	468	4.21
5	454	6.63	482	10.69	358	26.36	13	0.09
6	272	4.77	472	14.79	480	21.68	438	4.12
7	475	4.49	479	15.86	478	19.85	482	6.38
8	478	4.18	447	13.53	478	18.02	482	3.24
9	458	5.15	476	49.03	469	20.93	94	1.12
10	476	5.85	453	16.62	482	34.04	459	3.6
11	393	10.25	482	21	480	25.16	466	4.65
12	332	5.72	482	17.05	474	38.09	478	3.86
13	476	7.25	451	18.98	465	34.05	438	1.74
14	482	7.94	482	12.27	462	24.54	428	5.24
15	476	7.53	479	14.6	482	17.09	13	0.08
16	474	6.78	459	16.09	458	21.07	432	4.32
17	449	24.34	478	17.5	426	26.04	23	0.21
18	482	5.73	480	17.22	480	43.36	447	2.09
19	474	9	449	13.42	480	19.85	443	1.15
20	114	3.17	376	16.44	473	24.95	469	4.67
avg	421.15	7.6995	463.7	17.414	467.5	26.3205	365	3.3535

Obviously, when the initial safe call doubled the step number and spend time are grow up. And when the initial safe cell to triple, the step number is not rise such obvious but the time do, I guess it is because matching spending too much time but most of information is not necessary.

About 0.5 case, the step number are also reducing extreme due to not enough information, think about the safe cell are disperse uniform and no hints are 0 then the game must be stuck.

I have learned:

First is about to solve propositional logic, and how much penalty on checking and matching those logics sentence. And how the strategy of the programing skill and logic about solving mine sweeper

remaining questions and ideas of future investigation.

In the second case I tested, the more information could make the program run more successfully.   But it also spends too much time to matching not necessary sentence. Maybe the program can be design to only when the knowledge has few single-lateral clauses. I used to try only when there is no any single-lateral start to matching, but the result wasn’t looked great.

And there are some videos on internet show that they can solve many game in short time, compare to them, 2 min is sort of slow. Though hardware is one need to consider, but my program still need to improve.

Extra:

1. How to use first-order logic here?

Universal and Existential Quantifier can use for making some rule like:

∀x IsNotMine(x) → (SumOfMine(Surround(x)) = hint(x))

Or

∀x HadDiscovered(IsMine(x)) → Win()

2. Discuss whether forward chaining or backward chaining applicable to this problem.

In some case that we can’t confirm to choose what position or the penalty is too huge we can use the way to guess. In normal case, I don’t think it can work because with less hints may make the player who choose with the way lose.

3. Propose some ideas about how to improve the success rate of "guessing" when you want to proceed from a "stuck" game.

I think there are a way is using forward chaining or backward chaining to compute all of possible end game situation and to compute what the position have the highest probability being safe.

4. Discuss ideas of modifying the method in Assignment#2 to solve the current problem.

Using MRV to choose those have only one possible value can assign and every time it chooses it would ask for the hint, then recalculate the MRV and reply the step. When the game stuck, using 3. referred way to guess new position.

Code:

makefile

```
all:
	g++ -O3 main.cpp mineSweeper.cpp -o mineSweeper –g
```

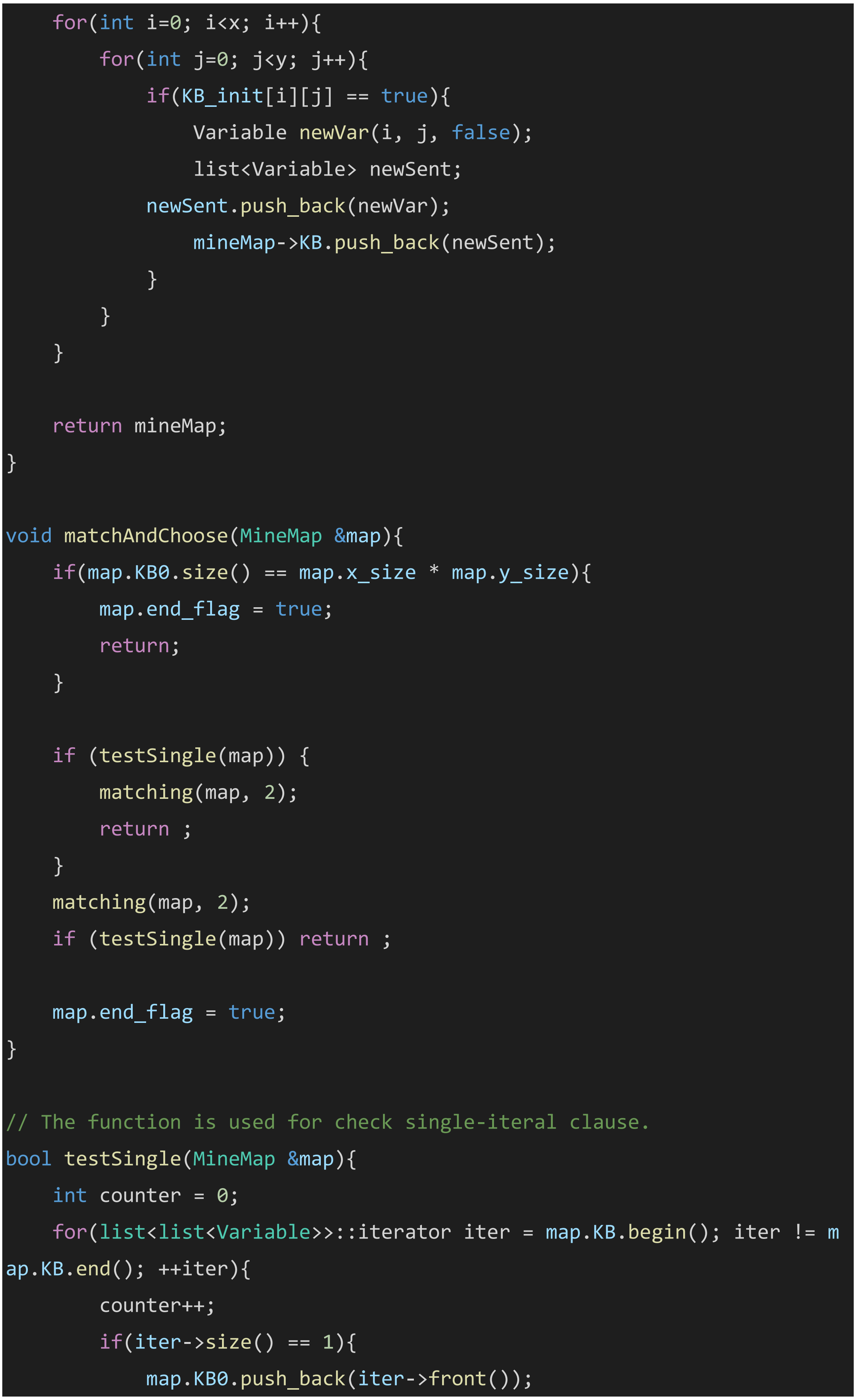
main.cpp

```cpp
#include "mineSweeper.hpp"
using namespace std;

int main(int argc, char** argv){
    if(argc != 2) {
        cout << "Usage: ./mineSweeper <Difficulty>" << endl;
        exit(1);
    }
    MineMap* map = mapInit(atoi(argv[1]));
    MineMap& map_ref = *map;

    //timer
    clock_t timer = clock();

    int counter = 1;
    while(map_ref.end_flag != true){
        matchAndChoose(map_ref);
        update(map_ref);

        counter++;
    // Comment the if condition can make the program dump the gameply o
n screen.
        if(map_ref.end_flag){
            cout << "The "<< counter << " step:" << endl;
            screenDump(map_ref);
        }
    }

    // Store data to a file
    ofstream data_file ("data.txt", ios::out | ios::app);
    data_file << clock() - timer << " " << counter << endl;
    data_file.close();

    delete map;
    return 0;
}
```

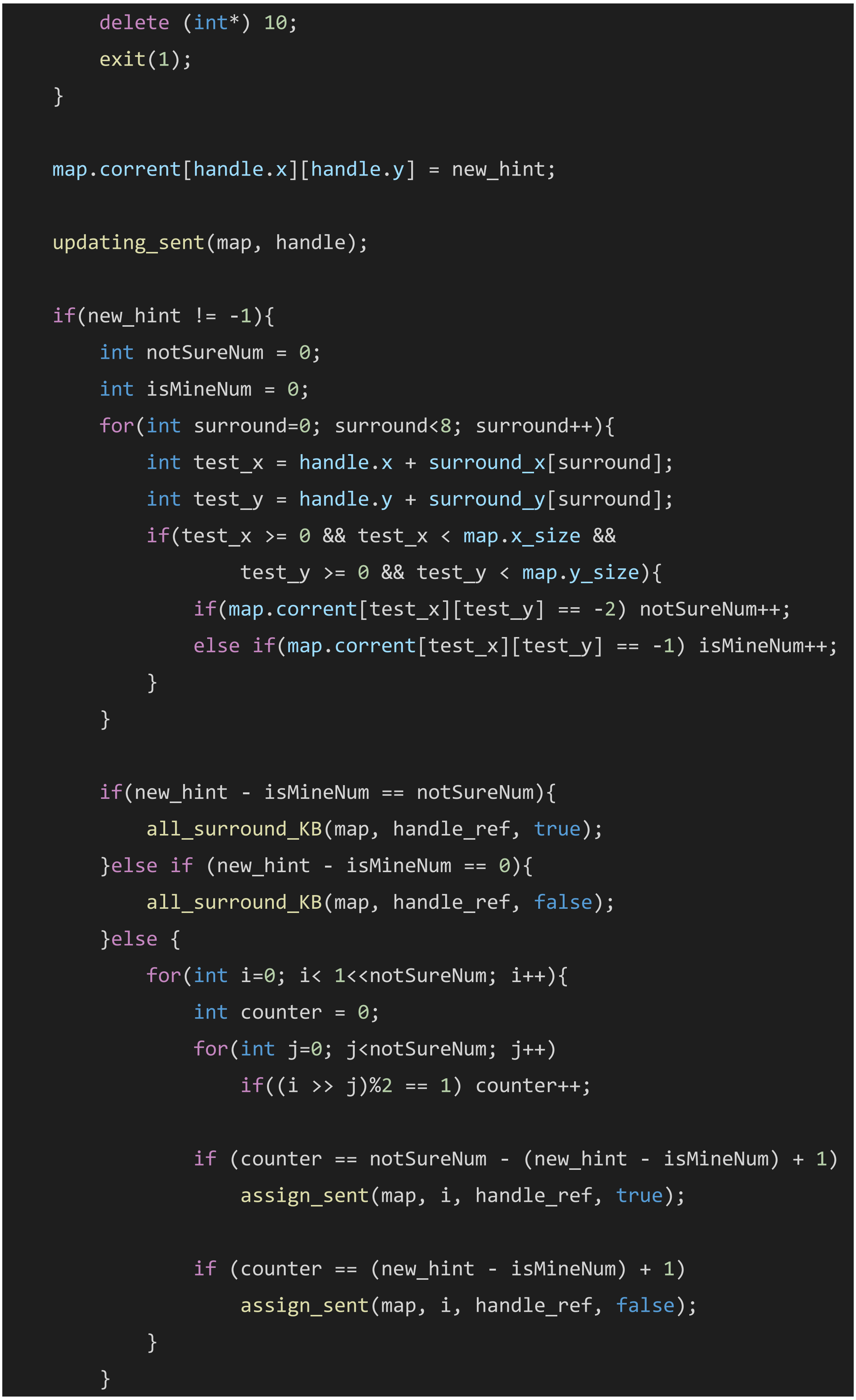
mineSweeper.cpp:

```cpp
#include "mineSweeper.hpp"
using namespace std;

//Initial the program, include:
//The difficulty,
//The real answer of the game,
//The initial safe cells to KB
//And also choose every mine and safe cells randomly.
MineMap* mapInit(int mode){
    int x, y, num;
    srand((unsigned) time(0));
    switch (mode){
        case 1:
            x = 9;
            y = 9;
            num = 10;
            break;
        case 2:
            x = 16;
            y = 16;
            num = 25;
            break;
        case 3:
            x = 16;
            y = 30;
            num = 99;
            break;
        case 4:
            cin >> x >> y >> num;
            break;
        default:
            exit(1);
    }
    MineMap* mineMap = new MineMap(x, y, num);

    mineMap->corrent.resize(x);
    mineMap->hints.resize(x);
    for(int i=0; i<x; i++){
        mineMap->corrent[i].resize(y, -2);
        mineMap->hints[i].resize(y, 0);
    }

    for (int mine=0; mine<num; mine++){
        int seq = rand() % (x * y - mine);
        int set_flag = 0;
        for(int i=0; i<x; i++){
            for(int j=0; j<y; j++){
                if(mineMap->hints[i][j] == -1) continue;
                else if (!seq) {
                    mineMap->hints[i][j] = -1;
                    set_flag = 1;
                    break;
                }
                else seq--;
            }
            if (set_flag) break;
        }
    }

    for(int i=0; i<x; i++){
        for(int j=0; j<y; j++){
            if(mineMap->hints[i][j] == -1) continue;
            for(int surround=0; surround<8; surround++){
                int testing_x = i + surround_x[surround];
                int testing_y = j + surround_y[surround];

                if(testing_x >= 0 && testing_x < x &&
                        testing_y >= 0 && testing_y < y &&
                        mineMap->hints[testing_x][testing_y] == -1
```

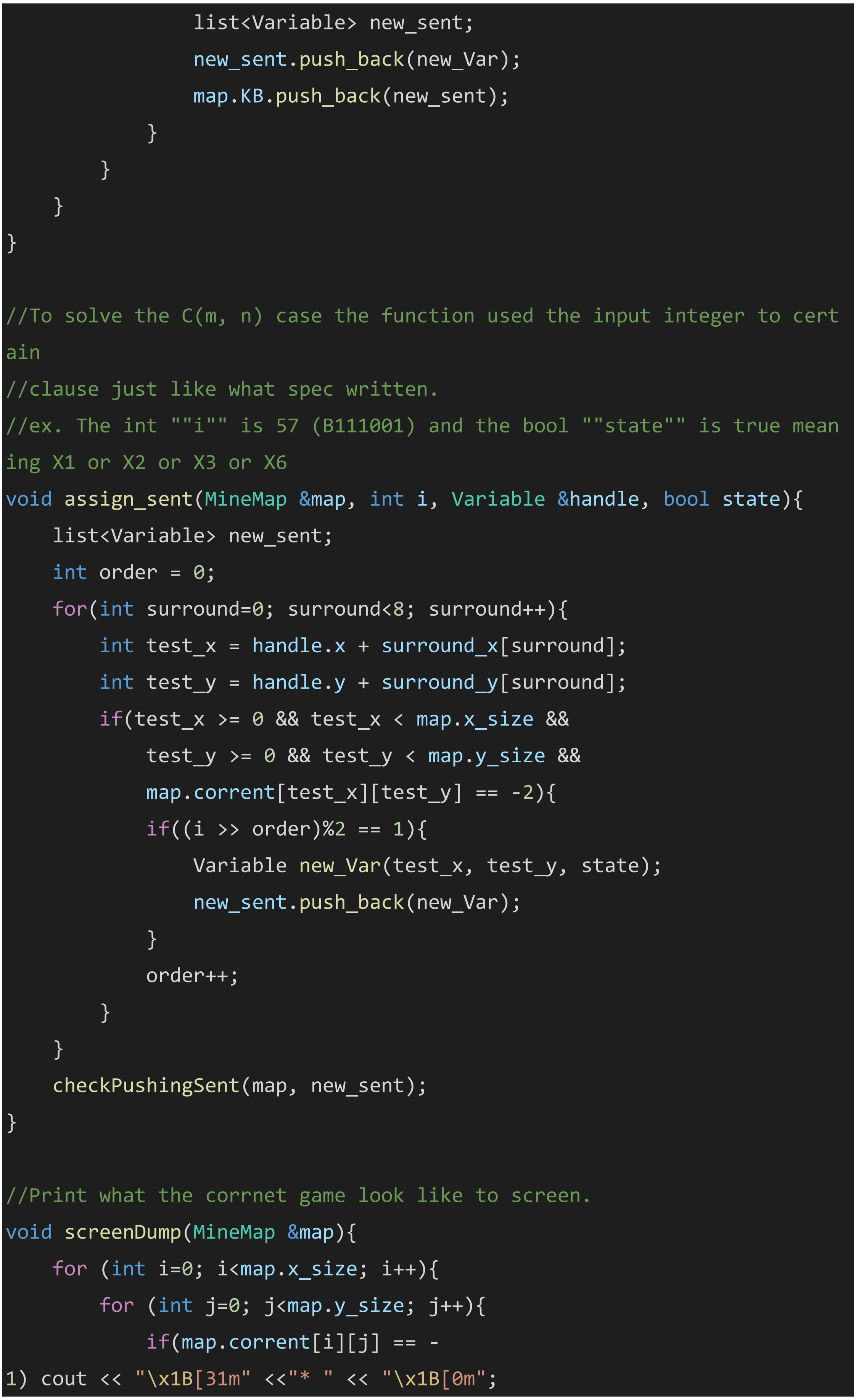
```
                  ) mineMap->hints[i][j]++;
            }
        }
    }

    cout << "Real answer:" << endl;
    for(int i=0; i<x; i++){
        for(int j=0; j<y; j++){
            if (mineMap->hints[i][j] == -1) cout << "* ";
            else cout << mineMap->hints[i][j] << " ";
        }
        cout << endl;
    }

    //here we try to modified the number
    int safe_init = round(sqrt(x*y));
    //safe_init/=2;
    //sleep(1);

    vector<vector<bool>> KB_init(x, vector<bool>(y, false));
    for (int safe=0; safe<safe_init; safe++){
        int seq = rand() % (x * y - num - safe);
        int set_flag = 0;
        for(int i=0; i<x; i++){
            for(int j=0; j<y; j++){
                if(mineMap->hints[i][j] == -
1 || KB_init[i][j] == true) continue;
                else if (!seq) {
                    KB_init[i][j] = true;
                    set_flag = 1;
                    break;
                }
                else seq--;
            }
            if (set_flag) break;
        }
    }
```



```
    for(int i=0; i<x; i++){
        for(int j=0; j<y; j++){
            if(KB_init[i][j] == true){
                Variable newVar(i, j, false);
                list<Variable> newSent;
            newSent.push_back(newVar);
                mineMap->KB.push_back(newSent);
            }
        }
    }

    return mineMap;
}

void matchAndChoose(MineMap &map){
    if(map.KB0.size() == map.x_size * map.y_size){
        map.end_flag = true;
        return;
    }

    if (testSingle(map)) {
        matching(map, 2);
        return ;
    }
    matching(map, 2);
    if (testSingle(map)) return ;

    map.end_flag = true;
}

// The function is used for check single-iteral clause.
bool testSingle(MineMap &map){
    int counter = 0;
    for(list<list<Variable>>::iterator iter = map.KB.begin(); iter != map.KB.end(); ++iter){
        counter++;
        if(iter->size() == 1){
            map.KB0.push_back(iter->front());
            map.KB.erase(iter);
            return true;
        }
    }
    return false;
}

//The function is used for resolvation
//The retunr and the mode is used for test differnt matching way
void matching(MineMap &map, int mode = 1){
//  return ;
    // Search two clauses which we need then do the resolvation
    for (list<list<Variable>>::iterator iter_i = map.KB.begin(); iter_i != map.KB.end(); ++iter_i ){
        for (list<list<Variable>>::iterator iter_j = next(iter_i); iter_j != map.KB.end(); ++iter_j ){
            if (mode == 1) {if(iter_i->size() != 2 || iter_j->size() != 2) continue;}
            else if(mode == 2) {if(iter_i->size() != 2 && iter_j->size() != 2) continue;}

            list<Variable>::iterator iter_var1 = iter_i->begin();
            list<Variable>::iterator iter_var2 = iter_j->begin();
            int pair = 0;
            Variable same_var;
            while(iter_var1 != iter_i->end() && iter_var2 != iter_j->end()){
                int tmp = compareVar(*iter_var1, *iter_var2);
                if(tmp == 0){
                    if(iter_var1->state != iter_var2->state){
                        pair ++;
                        same_var = *iter_var1;
                    }
                    iter_var1++;
                    iter_var2++;
                }else if(tmp == 1) iter_var2++;
                else iter_var1++;
            }

            // Test only with one pair between two clause
            if(pair != 1) continue;

            iter_var1 = iter_i->begin();
            iter_var2 = iter_j->begin();

            list<Variable> new_sent;
            while(iter_var1 != iter_i->end() && iter_var2 != iter_j-
>end()){
                int tmp = compareVar(*iter_var1, *iter_var2);
                if(tmp == 0){
                    if(iter_var1->x != same_var.x || iter_var1-
>y != same_var.y || iter_var1->state != same_var.state){
                        new_sent.push_back(*iter_var1);
                    }
                    iter_var1++;
                    iter_var2++;
                }else if(tmp == 1){
                    new_sent.push_back(*iter_var2);
                    iter_var2++;
                }else {
                    new_sent.push_back(*iter_var1);
                    iter_var1++;
                }
            }
            while(iter_var1 != iter_i->end()){
                new_sent.push_back(*iter_var1);
                iter_var1++;
            }
            while(iter_var2 != iter_j->end()){
                new_sent.push_back(*iter_var2);
                iter_var2++;
            }

            list<list<Variable>>::iterator& safe = iter_i;
            if(checkPushingSent_s(map, new_sent, safe)) iter_j = iter_i;
            if(iter_i == map.KB.end()) return;
            if(iter_j == map.KB.end()) break;
        }
    }
}

// Easy function to compare two vairable with coordination.
//  1: v1 > v2;
//  0: v1 == v2;
// -1: v1 < v2;
inline int compareVar(Variable v1, Variable v2){
    if(v1.x > v2.x) return 1;
    else if(v1.x < v2.x) return -1;
    else {
        if(v1.y > v2.y) return 1;
        else if(v1.y < v2.y) return -1;
        else if (v1.y == v2.y) return 0;
    }
}

// Check there are any duplicate clause
// And also check about are there any stricter clause any delete the unless information
// And the ""safe"" parameter is used for the other function need.
bool checkPushingSent_s(MineMap& map,list<Variable> new_sent, list<list<Variable>>::iterator& safe){
    bool modified_flag = false;
    for (list<list<Variable>>::iterator iter_i = map.KB.begin(); iter_i != map.KB.end(); ){
        int pair = 0;
        list<Variable>::iterator iter_var1 = iter_i->begin();
        list<Variable>::iterator iter_var2 = new_sent.begin();

        while(iter_var1 != iter_i->end() && iter_var2 != new_sent.end()){
            int tmp = compareVar(*iter_var1, *iter_var2);
            if(tmp == 0){
                if(iter_var1->state == iter_var2->state){
                    pair ++;
                }
                iter_var1++;
                iter_var2++;
            }else if(tmp == 1) iter_var2++;
            else iter_var1++;
        }
        if(pair == iter_i->size()){
            return modified_flag;
        }else if(pair == new_sent.size()){
            if (safe == iter_i){
                safe++;
            }
            modified_flag = true;
            iter_i = map.KB.erase(iter_i);
        }else {
            iter_i++;
        }
    }
    map.KB.push_back(new_sent);
    return modified_flag;
}

// Similar to upper function but without ""safe"" paremeter
void checkPushingSent(MineMap& map,list<Variable> new_sent){
    for (list<list<Variable>>::iterator iter_i = map.KB.begin(); iter_i != map.KB.end(); ){
        int pair = 0;
        list<Variable>::iterator iter_var1 = iter_i->begin();
        list<Variable>::iterator iter_var2 = new_sent.begin();

        while(iter_var1 != iter_i->end() && iter_var2 != new_sent.end()){
            int tmp = compareVar(*iter_var1, *iter_var2);
            if(tmp == 0){
                if(iter_var1->state == iter_var2->state){
                    pair ++;
                }
                iter_var1++;
                iter_var2++;
            }else if(tmp == 1) iter_var2++;
            else iter_var1++;
        }
        if(pair == iter_i->size()){
            return ;
        }else if(pair == new_sent.size()){
            iter_i = map.KB.erase(iter_i);
        }else {
            iter_i++;
        }
    }
    map.KB.push_back(new_sent);
}

//adding clause
//Process the "matching" of that clause to all the remaining clauses in
 the KB.
//If new clauses are generated due to resolution, insert them into the
KB.
//If this cell is safe:
//Query the game control module for the hint at that cell.
//Insert the clauses regarding its unmarked neighbors into the KB""
void update(MineMap &map){
    Variable handle = map.KB0.back();
    Variable& handle_ref = handle;

    //// To make sure the query is causely
    int new_hint = map.hints[handle.x][handle.y];
    if((handle.state == true && new_hint != -
1) ||(handle.state == false && new_hint == -1)){
        //if program is correct, then it would not happen.
        cout << "*****BOOM*****" << endl;
```



```
        delete (int*) 10;
        exit(1);
    }

    map.corrent[handle.x][handle.y] = new_hint;

    updating_sent(map, handle);

    if(new_hint != -1){
        int notSureNum = 0;
        int isMineNum = 0;
        for(int surround=0; surround<8; surround++){
            int test_x = handle.x + surround_x[surround];
            int test_y = handle.y + surround_y[surround];
            if(test_x >= 0 && test_x < map.x_size &&
                    test_y >= 0 && test_y < map.y_size){
                if(map.corrent[test_x][test_y] == -2) notSureNum++;
                else if(map.corrent[test_x][test_y] == -1) isMineNum++;
            }
        }

        if(new_hint - isMineNum == notSureNum){
            all_surround_KB(map, handle_ref, true);
        }else if (new_hint - isMineNum == 0){
            all_surround_KB(map, handle_ref, false);
        }else {
            for(int i=0; i< 1<<notSureNum; i++){
                int counter = 0;
                for(int j=0; j<notSureNum; j++)
                    if((i >> j)%2 == 1) counter++;

                if (counter == notSureNum - (new_hint - isMineNum) + 1)
                    assign_sent(map, i, handle_ref, true);

                if (counter == (new_hint - isMineNum) + 1)
                    assign_sent(map, i, handle_ref, false);
            }
        }
    }

}

//After pushing a clase into KB0, delete every useless clause and reslo
ved the clause have negtive variable
void updating_sent(MineMap& map, Variable &handle){
    for(list<list<Variable>>::iterator iter_i = map.KB.begin(); iter_i
!= map.KB.end(); ){ //++ set in block
        bool sentInvaildFlag = false;
        for (list<Variable>::iterator iter_j = iter_i-
>begin(); iter_j != iter_i->end(); ){
            if(iter_j->x == handle.x && iter_j->y == handle.y){
                if(iter_j->state == handle.state){
                    sentInvaildFlag = true;
                    break;
                }else {
                    iter_j = iter_i->erase(iter_j);
                    if(iter_i->size() == 0){
                        sentInvaildFlag = true;
                    }
                }
            }else iter_j++;
        }
        if(sentInvaildFlag) iter_i = map.KB.erase(iter_i);
        else iter_i++;
    }
}

//push all of cell surround the targe cell to KB with the state
void all_surround_KB(MineMap &map, Variable &handle, bool state){
    for(int surround=0; surround<8; surround++){
        int test_x = handle.x + surround_x[surround];
        int test_y = handle.y + surround_y[surround];
        if(test_x >= 0 && test_x < map.x_size &&
                test_y >= 0 && test_y < map.y_size){
            if(map.corrent[test_x][test_y] == -2){
                Variable new_Var(test_x, test_y, state);
```



```cpp
                list<Variable> new_sent;
                new_sent.push_back(new_Var);
                map.KB.push_back(new_sent);
            }
        }
    }
}

//To solve the C(m, n) case the function used the input integer to certain
//clause just like what spec written.
//ex. The int ""i"" is 57 (B111001) and the bool ""state"" is true meaning X1 or X2 or X3 or X6
void assign_sent(MineMap &map, int i, Variable &handle, bool state){
    list<Variable> new_sent;
    int order = 0;
    for(int surround=0; surround<8; surround++){
        int test_x = handle.x + surround_x[surround];
        int test_y = handle.y + surround_y[surround];
        if(test_x >= 0 && test_x < map.x_size &&
            test_y >= 0 && test_y < map.y_size &&
            map.corrent[test_x][test_y] == -2){
            if((i >> order)%2 == 1){
                Variable new_Var(test_x, test_y, state);
                new_sent.push_back(new_Var);
            }
            order++;
        }
    }
    checkPushingSent(map, new_sent);
}

//Print what the corrnet game look like to screen.
void screenDump(MineMap &map){
    for (int i=0; i<map.x_size; i++){
        for (int j=0; j<map.y_size; j++){
            if(map.corrent[i][j] == -1) cout << "\x1B[31m" <<"* " << "\x1B[0m";
            else if(map.corrent[i][j] == -
2) cout  << "\x1B[92m" << ". " << "\x1B[0m";
            else if (map.corrent[i][j] == 0) cout << "\x1B[90m" << map.
corrent[i][j] << "\x1B[0m" << " ";
            else cout <<  map.corrent[i][j] << " ";
        }
        cout << endl;
    }
}
```

mineSweeper.hpp:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <list>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <unistd.h>
using namespace std;


// 1 2 3
// 4 x 5
// 6 7 8
const int surround_x[8] = {-1, -1, -1,  0,  0,  1,  1,  1};
const int surround_y[8] = {-1,  0,  1, -1,  1, -1,  0,  1};

class Variable;
class MineMap{
    public:
        int x_size;
        int y_size;
        int mine_num;

        vector<vector<int>> corrent;
        vector<vector<int>> hints;
        bool end_flag = false;

        list<Variable> KB0;
        list<list<Variable>> KB;

        MineMap(int x, int y, int num){
            x_size = x;
            y_size = y;
            mine_num = num;
        }
};

class Variable{
    public:
        int x, y;
        bool state;

        Variable(int input_x, int input_y, bool isNotNeg){
            x = input_x;
            y = input_y;
            state = isNotNeg;
        }
        Variable(){}

};

MineMap* mapInit(int);

void matchAndChoose(MineMap&);
bool testSingle(MineMap&);
void matching(MineMap&, int);
int compareVar(Variable, Variable);


void update(MineMap&);
void updating_sent(MineMap&, Variable &);
void all_surround_KB(MineMap &, Variable &, bool);
void assign_sent(MineMap &, int , Variable &, bool);
void checkPushingSent(MineMap&, list<Variable>);
```

```
bool checkPushingSent_s(MineMap&, list<Variable>, list<list<Variable>>::iterator&);

void screenDump(MineMap&);
```

test.sh

```
for test_num in {0..10}
do
    ./mineSweeper 3
done
```