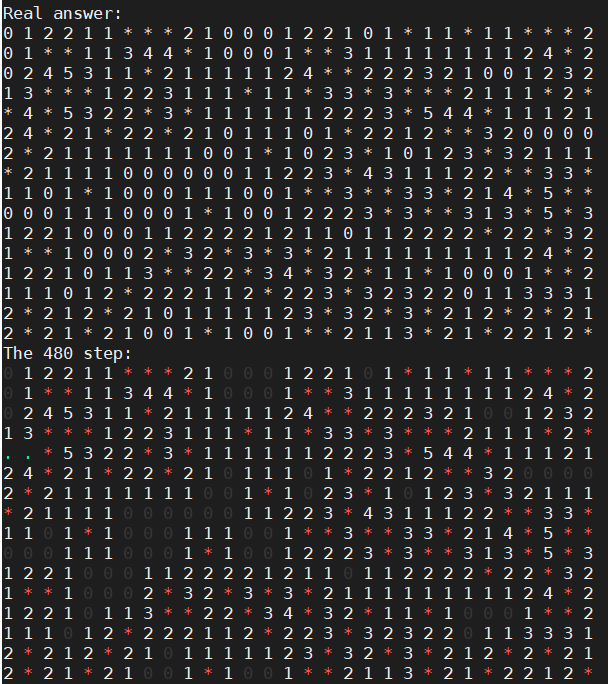
Programming Assignment #3

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) .

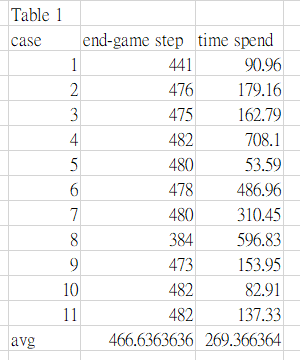


If you want to re-compile it, only need to input “make” and used the makedile

**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).

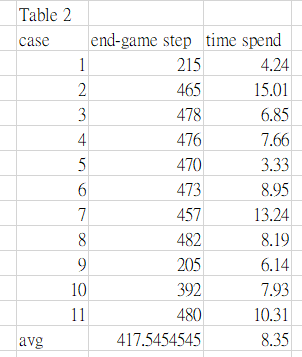
The table show that although halt of tests is not reach the end step, but they still complete most of game. And the average time it spends is about 2 min.



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.

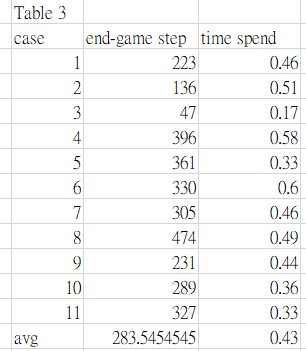
However, the time spending only about 8 second, reducing extremely.



Last, I want to try the case without any matching

In the following table (3), only one game arrives to more than 400 step.

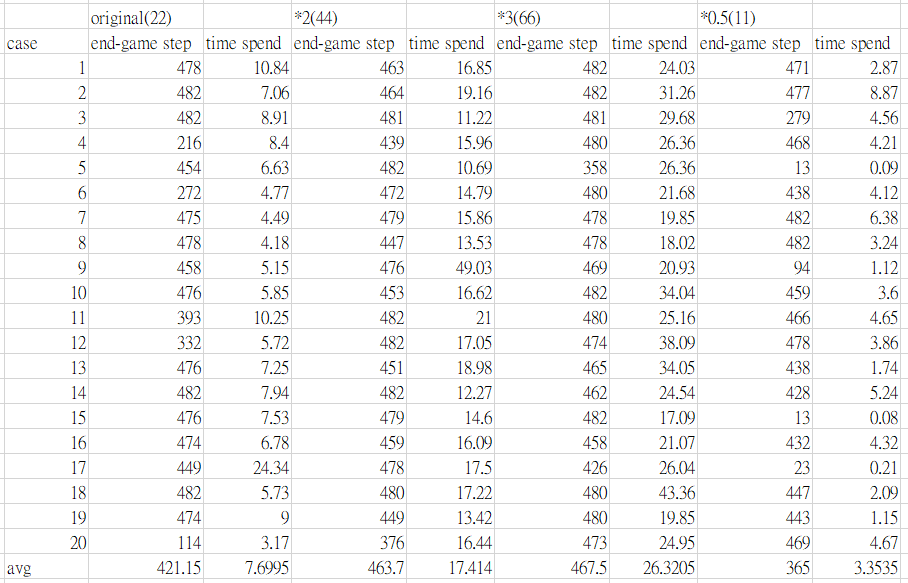
And average step also less than 300. Although one game only spends not more than 0.5 second, but it also useless.



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.



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()

1. 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.

1. 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.

1. 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***

#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 on 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:

#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 resloved 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);

                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