Programming Assignment #3

0611031 資工 10 謝至恆

**目標:**

寫出一個踩地雷的程式，可以自己產生一個有地雷的棋盤，並有

Easy 9\*9

Medium 16 \* 16

Hard 16 \* 30

3種模式，

還需要寫一個由全部空白棋盤用Propositional logic演算得到解答的程式

一開始會先給予round(sqrt(board\_size)個已經打開的提示來讓程是有條件可以演算

**解題思路:**

需要一個KB ，裡面包含single-lateral clause 代表一開始安全的提示，並從KB 中判斷每個格子中是安全或是有地雷。一值檢查剩餘的clauses 如果resolution產生新的clause也要加進KB裡面

一個KB0，一開始是空的，若KB判斷出格子是安全或有地雷，把這個位置和其bool值存到KB0。

如果判斷出一個格子是safe則要跟生成遊戲的module要那一格的提示，

每個cell有兩個狀態(safe)沒有地雷，(mined)有地雷，用bool來表示

遊戲結束 :

當所有格子都被打開或是被標記程地雷，且跟答案結果一致，代表成功

若最後剩下的格子有可能是地雷，則會無限跑一樣的迴圈，這時應該也算成功

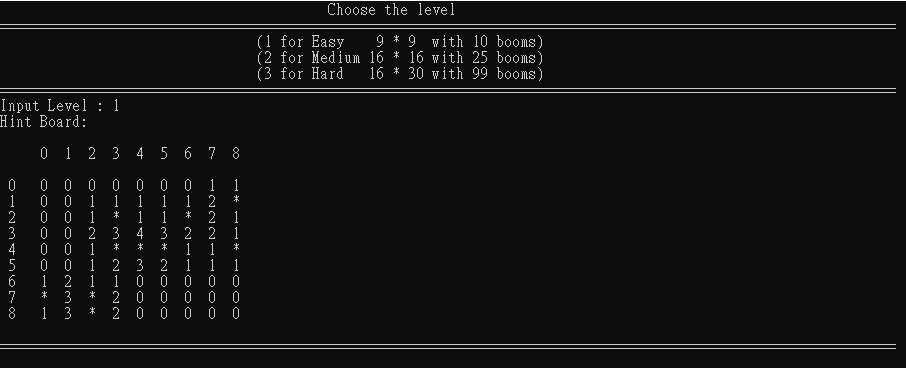
其他表示失敗。

**實作:**

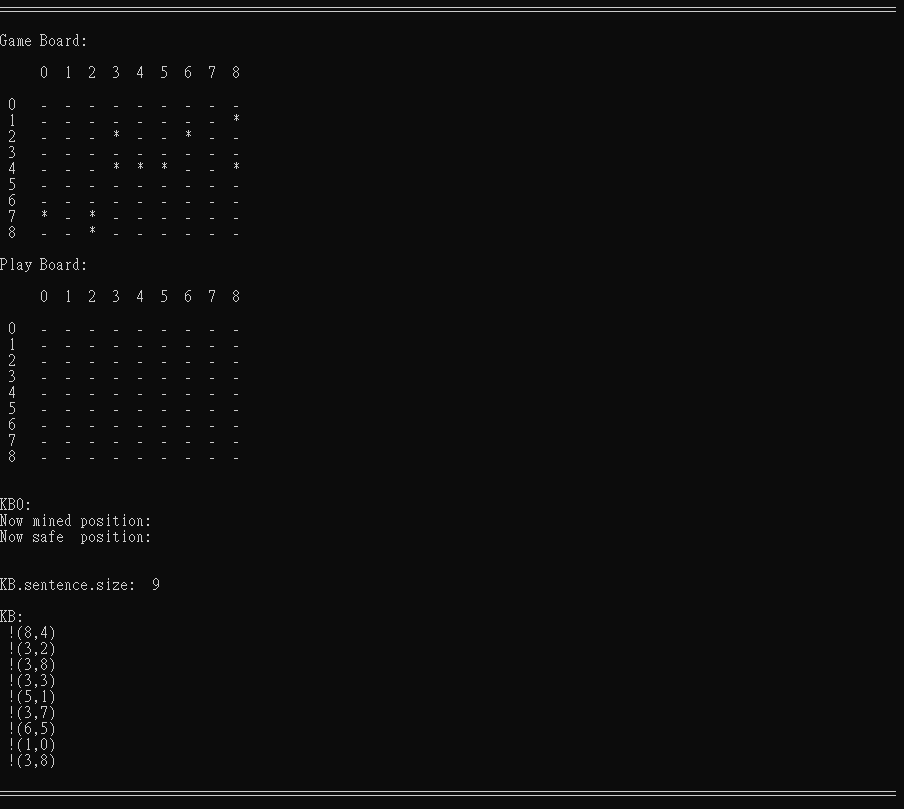
首先先生成實際的棋盤和玩家的棋盤，一個是知道所有安全位置和地雷(地雷是隨機的)，一個是全部未知的狀態，接著生成符合數目的提示，接下來就跟玩家的棋盤比較相關了，根據題是先產生相對應的clause，之後先把周圍地雷是0的先打開，也就隨之產生新的clause，這時候就要判斷有沒有可以resolution和subsumpiton的條件句，重複的clause和更嚴格的要把他刪除，才不會讓clause爆炸成長，打開一個提示後可以根據其條件判斷旁邊的棋盤能不能打開，如果打開也是安全的格子，就會多出一個hint，則要繼續根據這個hint再生成新的clause就這樣一值判斷，幸運的話一路到底就可以得到演算出來的棋盤，有時候會有解不出來的情況，也就是剩下的格子都有可能有地雷，這樣我們的演算法會一值跑相同的clause而且也沒辦法生成新的或消除clause，發現這個狀況的時候就要將成是停止，我是設定觀察KB的大小，如果跑100次都跑SIZE都沒有變化，就讓程式停止，結束這個棋盤

程式；

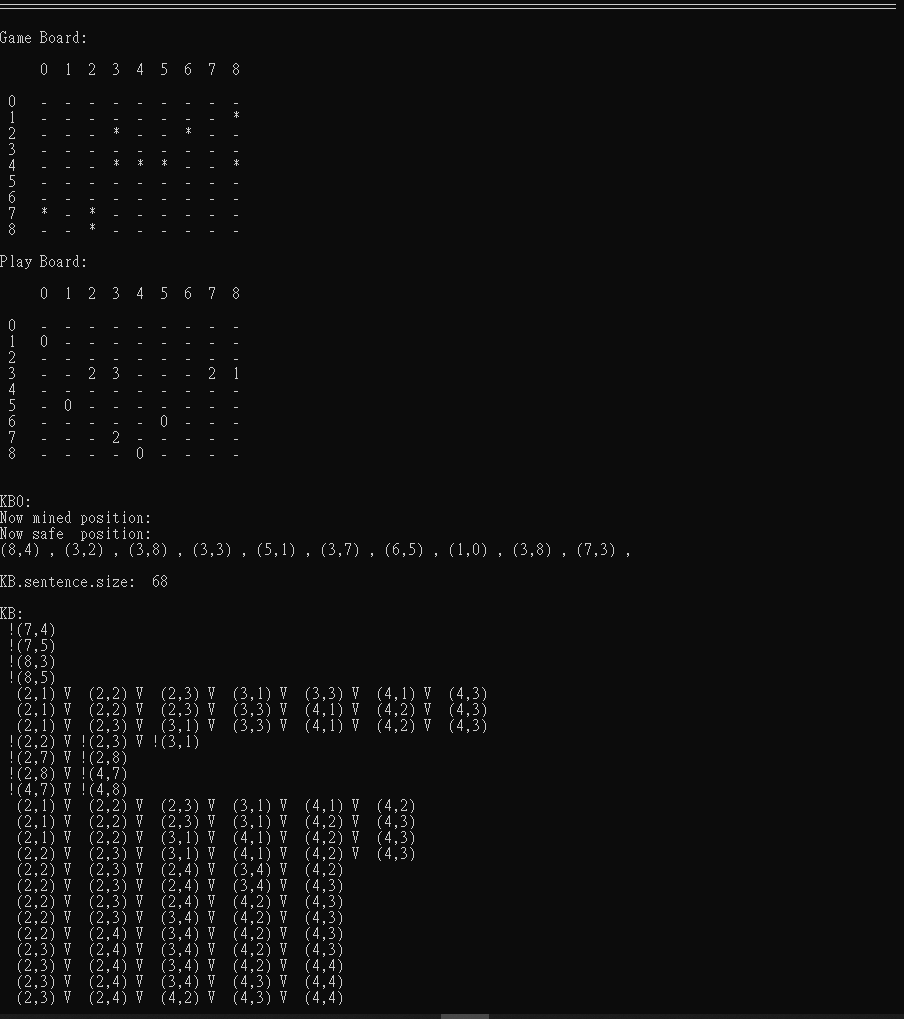
一開始先印出正確解答

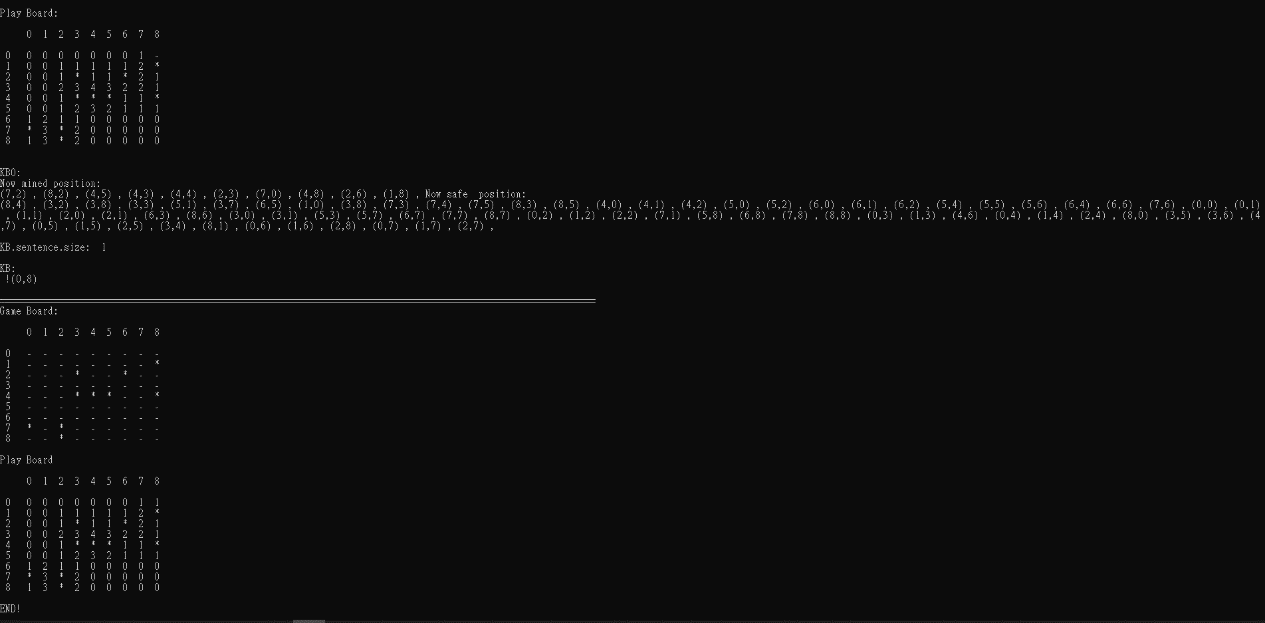


接下來是電腦看到的BOARD和玩家看到BOARD和最初的提示

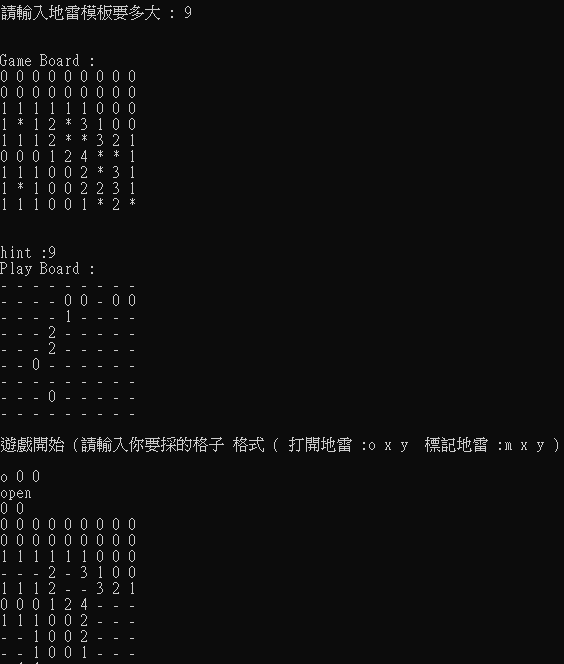


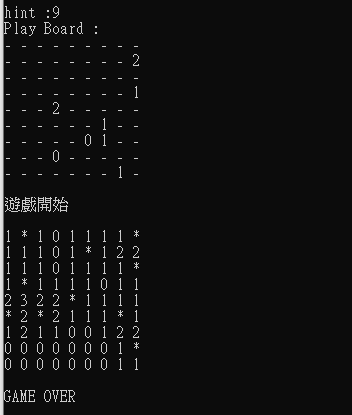
持續運算KnowledgeBase， KnowledgeBase0





這是驗算用的





**心得；**

這次的作業我花了很多時間在看懂作業的解釋和要求，不知道是不是我的問題，一開始看的時候，不太懂是要可以自己操作還是要直接跑出演算法算出來的遊玩結果，後來大概看了2~3遍加上問比較厲害的同學和討論才懂這次作業的架構，但就算搞懂後，實際做幾來才發現非常複雜，很多條件一判斷錯，就會一直跑無窮迴圈，這時候也很難DEBUG，只能把部分重來或是，逐行把現在再運算的條件都印出來。後來跟同學討論的時候發現PAIR這個函數可以把兩個資料連在一起，再使用Vector就可以得到很多類似資料結構的資料，這樣寫起來比物件更簡單，而且也更容易辨識，算是這次作業學到的東西最重要的吧。

感覺範例的部分可以多給一點線索或是有個明確的表示方法，比較不會一開始做一頭霧水，最後又砍掉重練，這次的作業自己寫起來感覺有點殺雞焉用牛刀的感覺，感覺要算踩地雷不需要做的這麼複雜，不過主要目的應該是了解propositional logic 的概念才對，這幾次作業下來我自己又多看了很多c++的函數庫，和複習很多寫法，也算是蠻有收穫的。

#include<bits/stdc++.h>

using namespace std;

//用pair來合併兩組資料　裡面放（是不是地雷, 位置）　vector來儲存所有pair資料

#define clause pair<bool,pair<int,int>>

#define position pair<int,int>

//X代表 pair的第一項 Y代表 pair的第二項

#define X first

#define Y second

//棋盤的全域變數

int BOARD\_HEIGHT\_X;

int BOARD\_WIDTH\_Y;

int BOOMS;

struct Board{

int width;

int height;

int booms;

vector<position> locate;

};

struct KnowledgeBase{

vector<vector<clause>> sentence;

};

KnowledgeBase KB;

struct KnowledgeBase0{

vector<position> safe;

vector<position> mined;

};

KnowledgeBase0 KB0;

vector<pair<int,int>> Init\_Safe\_Hint;

//生成一開始的棋盤

void generate\_board(int level, char game\_board[][30], char play\_board[][30],char hint\_board[][30]){

if(level == 1){

BOARD\_HEIGHT\_X = 9;

BOARD\_WIDTH\_Y = 9;

BOOMS = 10 ;

}

else if(level == 2){

BOARD\_HEIGHT\_X = 16;

BOARD\_WIDTH\_Y = 30;

BOOMS = 25;

}

else if(level == 2){

BOARD\_HEIGHT\_X = 16;

BOARD\_WIDTH\_Y = 30;

BOOMS = 25;

}

else{

cout << "Error input ! Please input a number range in 1 to 3" << endl;

return;

}

//生成棋盤

//產生全空的板子

for(int i=0;i<BOARD\_HEIGHT\_X;i++){

for(int j=0; j<BOARD\_WIDTH\_Y;j++){

game\_board[i][j]='-';

play\_board[i][j]='-';

hint\_board[i][j]='-';

}

}

srand(time(NULL));

//隨機放入地雷

for(int i=0;i<BOOMS;i++){

int x = rand()% BOARD\_HEIGHT\_X;

int y = rand()% BOARD\_WIDTH\_Y;

while(game\_board[x][y]!='-'){

x = rand()% BOARD\_HEIGHT\_X;

y = rand()% BOARD\_WIDTH\_Y;

}

game\_board[x][y]='\*';

hint\_board[x][y]='\*';

}

/\*//算出安全的地方周圍有幾個地雷

for(int i=0;i<BOARD\_HEIGHT\_X;i++){

for(int j=0;j<BOARD\_WIDTH\_Y;j++){

int number\_boom=0;

if( i>0 && j>0 && game\_board[i-1][j-1]=='\*'){

number\_boom++;

}

if( i>0 && game\_board[i-1][j]=='\*'){

number\_boom++;

}

if( i>0 && j+1<BOARD\_WIDTH\_Y && game\_board[i-1][j+1]=='\*'){

number\_boom++;

}

if( j>0 && game\_board[i][j-1]=='\*'){

number\_boom++;

}

if( j+1<BOARD\_WIDTH\_Y && game\_board[i][j+1]=='\*'){

number\_boom++;

}

if( i+1<BOARD\_HEIGHT\_X && j>0 && game\_board[i+1][j-1]=='\*'){

number\_boom++;

}

if( i+1<BOARD\_HEIGHT\_X && game\_board[i+1][j]=='\*'){

number\_boom++;

}

if( i+1<BOARD\_HEIGHT\_X && j+1<BOARD\_WIDTH\_Y && game\_board[i+1][j+1]=='\*'){

number\_boom++;

}

if(game\_board[i][j]!='\*'){

game\_board[i][j]=number\_boom+'0';

}

}

\*/

//隨機選定幾個安全點將位置存到pos裡，在將pos push進init\_safe裡面

int initial\_hints=round(sqrt(BOARD\_HEIGHT\_X\*BOARD\_WIDTH\_Y));

for(int i=0;i<initial\_hints;){

int random = rand() % (BOARD\_HEIGHT\_X\*BOARD\_WIDTH\_Y);

int x = random / BOARD\_HEIGHT\_X;

int y = random % BOARD\_WIDTH\_Y;

if(game\_board[x][y] != '\*'){

pair<int,int> pos = make\_pair(x,y);

Init\_Safe\_Hint.push\_back(pos);

i++;

}

}

}

//得到確定安全的座標的hint

void Hint\_Board(int level, char game\_board[][30],char hint\_board[][30]){

if(level == 1){

BOARD\_HEIGHT\_X = 9;

BOARD\_WIDTH\_Y = 9;

BOOMS = 10 ;

}

else if(level == 2){

BOARD\_HEIGHT\_X = 16;

BOARD\_WIDTH\_Y = 30;

BOOMS = 25;

}

else if(level == 2){

BOARD\_HEIGHT\_X = 16;

BOARD\_WIDTH\_Y = 30;

BOOMS = 25;

}

//算出安全的地方周圍有幾個地雷

for(int i=0;i<BOARD\_HEIGHT\_X;i++){

for(int j=0;j<BOARD\_WIDTH\_Y;j++){

int number\_boom=0;

if( i>0 && j>0 && game\_board[i-1][j-1]=='\*'){

number\_boom++;

}

if( i>0 && game\_board[i-1][j]=='\*'){

number\_boom++;

}

if( i>0 && j+1<BOARD\_WIDTH\_Y && game\_board[i-1][j+1]=='\*'){

number\_boom++;

}

if( j>0 && game\_board[i][j-1]=='\*'){

number\_boom++;

}

if( j+1<BOARD\_WIDTH\_Y && game\_board[i][j+1]=='\*'){

number\_boom++;

}

if( i+1<BOARD\_HEIGHT\_X && j>0 && game\_board[i+1][j-1]=='\*'){

number\_boom++;

}

if( i+1<BOARD\_HEIGHT\_X && game\_board[i+1][j]=='\*'){

number\_boom++;

}

if( i+1<BOARD\_HEIGHT\_X && j+1<BOARD\_WIDTH\_Y && game\_board[i+1][j+1]=='\*'){

number\_boom++;

}

if(hint\_board[i][j]!='\*'){

hint\_board[i][j]=number\_boom+'0';

}

}

}

}

//印出棋盤的座標位置

void PrintBoard(char game\_board[][30]){

cout << endl;

cout << " ";

for(int i = 0; i < BOARD\_WIDTH\_Y; i++){

cout << i << " ";

if(i < 10) cout << " ";

}

cout << endl<<endl ;

for(int i = 0; i < BOARD\_HEIGHT\_X; i++){

if(i < 10) cout << " ";

cout << i << " ";

for(int j = 0; j < BOARD\_WIDTH\_Y; j++){

cout << game\_board[i][j] << " ";

cout << " ";

}

cout << endl;

}

cout << endl;

}

bool subsumption(vector<clause> temp){

// cout << endl << "subsumption " <<endl;

bool condition=false;

vector<vector<clause>> newKB = KB.sentence;

bool erase = false;

//檢查兩個句子有多少相同

for(int i = 0; i < KB.sentence.size(); i++){

int number\_ident = 0;

for(int j = 0; j < KB.sentence[i].size(); j++){

for(int k = 0; k < temp.size(); k++){

if(temp[k].second.X == KB.sentence[i][j].second.X && temp[k].second.Y == KB.sentence[i][j].second.Y){

if(temp[k].first == KB.sentence[i][j].first){

number\_ident++;

}

}

}

}

if(KB.sentence[i].size() == temp.size()){

if(number\_ident == temp.size()){

//相同的sentence

if(erase) KB.sentence = newKB;

return true;

}

}

if(condition){

cout << "condition";

}

if(number\_ident == temp.size()){

cout << "\nerase " << i << endl;

for(int j = 0; j < KB.sentence[i].size(); j++){

if(KB.sentence[i][j].first == false) cout << " !";

else cout << " ";

cout << "(" << KB.sentence[i][j].second.X << "," << KB.sentence[i][j].second.Y << ") ";

if(j < KB.sentence[i].size()) cout << "V";

}

//新的比較嚴格，刪掉舊的，將新的加入KB

auto k = find(begin(newKB), end(newKB), KB.sentence[i]);

newKB.erase(k);

erase = true;

cout << newKB.size() << endl;

}

if(number\_ident == KB.sentence[i].size()){//如果現在的比較嚴格，刪掉新的

if(erase) KB.sentence = newKB;

return true;

}

}

if(condition){

cout << "condition";

}

if(erase) KB.sentence = newKB;

return false;

}

//當有相同位置存在不同的bool值，可以做resolution來減少條件判斷

void resolution\_KB0(vector<clause> &temp){

bool earse=false;

// cout << endl << "resolution" <<endl;

for(int i = 0; i < KB0.safe.size(); i++){

for(int j = 0; j < temp.size(); j++){

if(KB0.safe[i].X == temp[j].second.X && KB0.safe[i].Y == temp[j].second.Y){

if(temp[j].first == true){

//不同號則刪掉

auto k = find(begin(temp), end(temp), temp[j]);

temp.erase(k);

}

else{

//同號就清掉就好

temp.clear();

}

}

}

}

for(int i = 0; i < KB0.mined.size(); i++){

for(int j = 0; j < temp.size(); j++){

if(KB0.mined[i].X == temp[j].second.X && KB0.mined[i].Y == temp[j].second.Y){//same position

if(temp[j].first == false){//opposite sign

auto k = find(begin(temp), end(temp), temp[j]);

temp.erase(k);

}

else{

temp.clear();

}

}

}

}

}

//有提示進來的時候，需要生成新的clause，

/\*

About generating clauses from the hints:

Each hint provides the following information: There are n mines in a list of m unmarked cells.

(n == m): Insert the m single-literal positive clauses to the KB, one for each unmarked cell.

(n == 0): Insert the m single-literal negative clauses to the KB, one for each unmarked cell.

(m>n>0): General cases (need to generate CNF clauses and add them to the KB):

C(m, m-n+1) clauses, each having m-n+1 positive literals

C(m, n+1) clauses, each having n+1 negative literals.

For example, for m=5 and n=2, let the cells be x1, x2, …, x5:

There are C(5,4) all-positive-literal clauses:

(x1 V x2 V x3 V x4), (x1 V x2 V x3 V x5), …, (x2 V x3 V x4 V x5)

There are C(5,3) all-negative-literal clauses:

(!x1 V !x2 V !x3), (!x1 V ! x2 V !x4), (!x1 V !x2 V !x5), …, (!x3 V !x4 V !x5)

\*/

void generate\_Clauses(int x, int y, int hint, char play\_board[][30], char hint\_board[][30]){

bool earse=false;

int mines = hint;

int unmarked = 0;

vector<position> list\_unmarked;

for(int i = x-1; i <= x+1; i++){

for(int j = y-1; j <= y+1; j++){

if((i >= 0) && (i < BOARD\_HEIGHT\_X) && (j >= 0) && (j < BOARD\_WIDTH\_Y)){

if(play\_board[i][j] == '-'){

unmarked++;

position pos = make\_pair(i,j);

list\_unmarked.push\_back(pos);

}

else if(play\_board[i][j] == '\*'){

mines--;

}

}

}

}

if(mines == unmarked){

// Insert the m single-literal positive clauses to the KB, one for each unmarked cell.

for(int i = x-1; i <= x+1; i++){

for(int j = y-1; j <= y+1; j++){

if((i >= 0) && (i < BOARD\_HEIGHT\_X) && (j >= 0) && (j < BOARD\_WIDTH\_Y)){

if(play\_board[i][j] == '-'){

position pos = make\_pair(i, j);

clause temp = make\_pair(true, pos);

vector<clause> temp\_clauses;

temp\_clauses.push\_back(temp);

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size())

KB.sentence.push\_back(temp\_clauses);

}

}

}

}

return;

}

if(mines == 0){

// Insert the m single-literal negative clauses to the KB, one for each unmarked cell.

for(int i = x-1; i <= x+1; i++){

for(int j = y-1; j <= y+1; j++){

if((i >= 0) && (i < BOARD\_HEIGHT\_X) && (j >= 0) && (j < BOARD\_WIDTH\_Y)){

if(play\_board[i][j] == '-'){

position pos = make\_pair(i, j);

clause temp = make\_pair(false, pos);

vector<clause> temp\_clauses;

temp\_clauses.push\_back(temp);

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size())

KB.sentence.push\_back(temp\_clauses);

}

}

}

}

return;

}

if(unmarked > mines){

// C(m, m-n+1) clauses, each having m-n+1 positive literals

vector<clause> temp\_clauses;

vector<int> combination(unmarked, 0);

for(int i = 0; i < (unmarked-mines+1); i++){

combination[i] = 1;

}

for(int i = 0; i < combination.size(); i++){

if(combination[i] == 1){

position pos = list\_unmarked[i];

clause temp = make\_pair(true, pos);

temp\_clauses.push\_back(temp);

}

}

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size()){

KB.sentence.push\_back(temp\_clauses);

}

for(int i = 0; i < unmarked - 1; i++){

temp\_clauses.clear();

if(combination[i] == 1 && combination[i+1] == 0){

swap(combination[i], combination[i+1]);

sort(combination.begin(), combination.begin()+i);

for(int i = 0; i < combination.size(); i++){

if(combination[i] == 1){

position pos = list\_unmarked[i];

clause temp = make\_pair(true, pos);

temp\_clauses.push\_back(temp);

}

}

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size()){

KB.sentence.push\_back(temp\_clauses);

}

i=-1;

if(earse){

KB.sentence.push\_back(temp\_clauses);

}

}

}

temp\_clauses.clear();

// C(m, n+1) clauses, each having n+1 negative literals

for(int i = 0; i < unmarked; i++){

if(i < (mines+1))

combination[i] = 1;

else

combination[i] = 0;

}

for(int i = 0; i < combination.size(); i++){

if(combination[i] == 1){

position pos = list\_unmarked[i];

clause temp = make\_pair(false, pos);

temp\_clauses.push\_back(temp);

}

}

if(earse){

KB.sentence.push\_back(temp\_clauses);

}

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size()){

KB.sentence.push\_back(temp\_clauses);

}

for(int i = 0; i < unmarked - 1; i++){

temp\_clauses.clear();

if(combination[i] == 1 && combination[i+1] == 0){

swap(combination[i], combination[i+1]);

sort(combination.begin(), combination.begin()+i);

for(int i = 0; i < combination.size(); i++){

if(combination[i] == 1){

position pos = list\_unmarked[i];

clause temp = make\_pair(false, pos);

temp\_clauses.push\_back(temp);

}

}

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size()){

KB.sentence.push\_back(temp\_clauses);

}

i=-1;

if(earse){

KB.sentence.push\_back(temp\_clauses);

}

}

}

}

}

//控制遊戲開始與結束

void GameStart(char game\_board[][30], char play\_board[][30],char hint\_board[][30]){

//將Init\_Safe\_Hint裡面所有位置(確定是沒有地雷)生成最一開始的clause

bool game\_end=false;

//check hint\_board

cout << "Hint Board:"<<endl;

cout << endl;

cout << " ";

for(int i = 0; i < BOARD\_WIDTH\_Y; i++){

cout << i << " ";

if(i < 10) cout << " ";

}

cout << endl<<endl ;

for(int i = 0; i < BOARD\_HEIGHT\_X; i++){

if(i < 10) cout << " ";

cout << i << " ";

for(int j = 0; j < BOARD\_WIDTH\_Y; j++){

cout << hint\_board[i][j] << " ";

cout << " ";

}

cout << endl;

}

cout << endl;

while(Init\_Safe\_Hint.size()){

position pos = Init\_Safe\_Hint.back();

Init\_Safe\_Hint.pop\_back();

clause temp = make\_pair(false, pos);

//生成一個vector裡面存一連串的clause的類型資料裡面存的是sentence，push\_back() - 新增元素至 vector 的尾端，必要時會進行記憶體配置。

vector<clause> temp\_clauses;

temp\_clauses.push\_back(temp);

KB.sentence.push\_back(temp\_clauses);

}

//當KB面還有sentence (則需要繼續檢查有沒有可以做的)

int identical=0;

int size=0;

int endn=0;

while(KB.sentence.size()&& endn==0){

if(KB.sentence.size()==size){

identical++;

}

if(identical == 100){

endn=1;

}

//每次都先印出現在棋盤和有的條件

cout << "================================================================================================================\n";

cout << endl;

cout << "Game Board:"<<endl;

PrintBoard(game\_board);

cout << "Play Board:"<<endl;

PrintBoard(play\_board);

cout << endl;

cout << "KB0:" << endl;

cout << "Now mined position:" << endl;

for(int i=0; i<KB0.mined.size();i++){

cout << "(" << KB0.mined[i].X << "," << KB0.mined[i].Y << ") , ";

}

cout << endl;

cout << "Now safe position:" << endl;

for(int i=0; i<KB0.safe.size();i++){

cout << "(" << KB0.safe[i].X << "," << KB0.safe[i].Y << ") , ";

}

cout << endl<<endl;

//檢查KB KB[i]但代表第幾個clause [i][j]代表裡面存的位置和bool值

cout << "KB.sentence.size: " << KB.sentence.size() << endl<<endl;

cout << "KB:"<<endl;

for(int i=0;i < KB.sentence.size();i++){

for(int j=0;j<KB.sentence[i].size(); j++){

if(KB.sentence[i][j].first == false) cout << " !";

else cout << " ";

cout << "(" << KB.sentence[i][j].second.X << "," << KB.sentence[i][j].second.Y << ") ";

if(j < KB.sentence[i].size()) cout << "V";

}

cout << endl;

}

cout << endl;

//如果有一個clause只有一個元素，代表可以確定她是地雷或是沒有(一次只做一個)

//則消除這個clause 並更新play\_board的值和放入KB0相應的地方

bool single = false;

for(int i = 0; i < KB.sentence.size(); i++){

if(KB.sentence[i].size() == 1){

single = true;

bool sign = KB.sentence[i][0].first;

position pos = KB.sentence[i][0].second;

auto e = find(begin(KB.sentence), end(KB.sentence), KB.sentence[i]);

KB.sentence.erase(e);

if(sign){

KB0.mined.push\_back(pos);

//標記是個地雷

//cout << hint\_board[pos.X][pos.Y] << endl; //檢查是不是真的是地雷

play\_board[pos.X][pos.Y] = '\*';

}

else{

KB0.safe.push\_back(pos);

//得到的提示加到play\_board

//cout << hint\_board[pos.X][pos.Y]<< endl; // 查看值

play\_board[pos.X][pos.Y] = hint\_board[pos.X][pos.Y];

}

//產生一個clause vector 存取這個點的clause值

clause temp = make\_pair(sign, pos);

//產生一個可以存clause　的 vector

vector<clause> temp\_clauses;

temp\_clauses.push\_back(temp);

//檢查這個\_clause會不會產生 subsumption (存在2 clause 一個是另外一個的子集合

//會刪除較不嚴格的那個

if(subsumption(temp\_clauses)){//Check for duplication or subsumption first. Keep only the more strict clause.

}

bool street;

for(int i = 0; i < KB.sentence.size(); i++){

for(int j = 0; j < KB.sentence[i].size(); j++){

if(pos.X == KB.sentence[i][j].second.X && pos.Y == KB.sentence[i][j].second.Y && sign != KB.sentence[i][j].first){

street = true;

}

}

}

//檢查這個新產生的clause會不會resolution，如果有將結果加進KB裡

for(int i = 0; i < KB.sentence.size(); i++){

for(int j = 0; j < KB.sentence[i].size(); j++){

if(pos.X == KB.sentence[i][j].second.X && pos.Y == KB.sentence[i][j].second.Y && sign != KB.sentence[i][j].first){

cout << endl << endl;

for(int m = 0; m < KB.sentence[i].size(); m++){

if(KB.sentence[i][m].first == false) cout << " !";

else cout << " ";

cout << "(" << KB.sentence[i][m].second.X << "," << KB.sentence[i][m].second.Y << ") ";

if(m < KB.sentence[i].size()) cout << "V";

}

//刪除resolution 的那個條件

auto k = find(begin(KB.sentence[i]), end(KB.sentence[i]), KB.sentence[i][j]);

KB.sentence[i].erase(k);

vector<clause> temp\_clauses = KB.sentence[i];

auto l = find(begin(KB.sentence), end(KB.sentence), KB.sentence[i]);

KB.sentence.erase(l);

resolution\_KB0(temp\_clauses);

for(int m = 0; m < temp\_clauses.size(); m++){

if(temp\_clauses[m].first == false) cout << " !";

else cout << " ";

cout << "(" << temp\_clauses[m].second.X << "," << temp\_clauses[m].second.Y << ") ";

if(m < temp\_clauses.size()) cout << "v";

}

if(subsumption(temp\_clauses) == false && temp\_clauses.size()){

KB.sentence.push\_back(temp\_clauses);

cout << "sub false\n";

}

}

}

}

//如果不是地雷

if(!sign){

int hint = hint\_board[pos.X][pos.Y]-'0';

//產生根據其hint周圍地雷數相對旁邊８個格子的clauses

generate\_Clauses(pos.X, pos.Y, hint, play\_board, hint\_board);

}

for(int i = 0; i < KB.sentence.size(); i++){

for(int j = 0; j < KB.sentence[i].size(); j++){

if(pos.X == KB.sentence[i][j].second.X && pos.Y == KB.sentence[i][j].second.Y && sign != KB.sentence[i][j].first){

street = true;

}

}

}

break;

}

}

if(single) continue;

//如果不是單一條件

else{

for(int k = 0; k < KB.sentence.size(); k++){

if(KB.sentence[k].size() > 2) continue;

bool sign0 = KB.sentence[k][0].first;

position pos0 = KB.sentence[k][0].second;

bool sign1 = KB.sentence[k][1].first;

position pos1 = KB.sentence[k][1].second;

for(int i = 0; i < KB.sentence.size(); i++){

int pairs = 0;

if(i == k) continue;

for(int j = 0; j < KB.sentence[i].size(); j++){

if(pos0.X == KB.sentence[i][j].second.X && pos0.Y == KB.sentence[i][j].second.Y && sign0 != KB.sentence[i][j].first)

pairs++;

if(pos1.X == KB.sentence[i][j].second.X && pos1.Y == KB.sentence[i][j].second.Y && sign1 != KB.sentence[i][j].first)

pairs++;

}

if(pairs == 1){

vector<clause> temp\_clauses;

//檢查這個新產生的clause會不會resolution，如果有將結果加進KB裡

for(int j = 0; j < KB.sentence[i].size(); j++){

if(pos0.X == KB.sentence[i][j].second.X && pos0.Y == KB.sentence[i][j].second.Y && sign0 != KB.sentence[i][j].first){

auto m = find(begin(KB.sentence[i]), end(KB.sentence[i]), KB.sentence[i][j]);

KB.sentence[i].erase(m);

clause temp = make\_pair(sign1, pos1);

KB.sentence[i].push\_back(temp);

temp\_clauses = KB.sentence[i];

auto l = find(begin(KB.sentence), end(KB.sentence), KB.sentence[i]);

KB.sentence.erase(l);

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size())

KB.sentence.push\_back(temp\_clauses);

}

//檢查這個新產生的clause會不會resolution，如果有將結果加進KB裡

else if(pos1.X == KB.sentence[i][j].second.X && pos1.Y == KB.sentence[i][j].second.Y && sign1 != KB.sentence[i][j].first){

auto m = find(begin(KB.sentence[i]), end(KB.sentence[i]), KB.sentence[i][j]);

KB.sentence[i].erase(m);

clause temp = make\_pair(sign0, pos0);

KB.sentence[i].push\_back(temp);

temp\_clauses = KB.sentence[i];

auto l = find(begin(KB.sentence), end(KB.sentence), KB.sentence[i]);

KB.sentence.erase(l);

resolution\_KB0(temp\_clauses);

if(subsumption(temp\_clauses) == false && temp\_clauses.size())

KB.sentence.push\_back(temp\_clauses);

}

}

}

}

}

}

size=KB.sentence.size();

}

cout <<"================================================================================================================\n";

cout << "Game Board:\n";

PrintBoard(game\_board);

cout << "Play Board\n";

PrintBoard(play\_board);

cout << "END!\n";

}

int main(){

int level;

while(1){

//初始化

Init\_Safe\_Hint.clear();

KB0.safe.clear();

KB0.mined.clear();

cout<<endl;

cout<<" Choose the level"<<endl;

cout<<"================================================================================================================\n";

cout<< " (1 for Easy 9 \* 9 with 10 booms)" <<endl;

cout<< " (2 for Medium 16 \* 16 with 25 booms)" <<endl;

cout<< " (3 for Hard 16 \* 30 with 99 booms)" <<endl;

cout<<"================================================================================================================\n";

cout<< "Input Level : ";

cin >> level;

char game\_board[30][30];

char play\_board[30][30];

char hint\_board[30][30];

generate\_board(level,game\_board,play\_board,hint\_board);

Hint\_Board(level,game\_board,hint\_board);

//PrintBoard(game\_board);

//PrintBoard(play\_board);

GameStart(game\_board,play\_board,hint\_board);

//Start(game\_board,play\_board);

}

return 0;

}

另一個版本 驗算用 暴力解

#include <iostream>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include <cstring>

using namespace std;

void generate\_board(char \*board, int height, int width){

int SquareN=round(sqrt(height\*width)+1);

//產生全空的板子

for(int i=0;i<height;i++){

for(int j=0; j<width;j++){

\*(board+i\*width+j)='-';

}

}

//隨機添加地雷

for(int i=0;i<SquareN;i++){

int x = rand()%height;

int y = rand()%height;

//cout << x << y <<endl;

while(\*(board+x\*width+y)!='-'){

x = rand()%height;

y = rand()%height;

}

\*(board+x\*width+y)='\*';

}

for(int i=0;i<height;i++){

for(int j=0;j<width;j++){

int number\_boom=0;

if( i>0 && j>0 && \*(board+(i-1)\*width+j-1)=='\*'){

number\_boom++;

//\*(board+(i-1)\*width+j-1)

//board[i-1][j-1]

}

if( i>0 && \*(board+(i-1)\*width+j)=='\*'){

number\_boom++;

//\*(board+(i-1)\*width+j)

//board[i-1][j]

}

if( i>0 && j+1<width && \*(board+(i-1)\*width+j+1)=='\*'){

number\_boom++;

//\*(board+(i-1)\*width+j+1)

//board[i-1][j+1]

}

if( j>0 && \*(board+i\*width+j-1)=='\*'){

number\_boom++;

//\*(board+i\*width+j-1)

//board[i][j-1]

}

if( j+1<width && \*(board+i\*width+j+1)=='\*'){

number\_boom++;

//\*(board+i\*width+j+1)

//board[i][j+1]

}

if( i+1<height && j>0 && \*(board+(i+1)\*width+j-1)=='\*'){

number\_boom++;

//\*(board+(i+1)\*width+j-1)

//board[i+1][j-1]

}

if( i+1<height && \*(board+(i+1)\*width+j)=='\*'){

number\_boom++;

//\*(board+(i+1)\*width+j)

//board[i+1][j]

}

if( i+1<height && j+1<width && \*(board+(i+1)\*width+j+1)=='\*'){

number\_boom++;

//\*(board+(i+1)\*width+j+1)

//board[i+1][j+1]

}

if(\*(board+i\*width+j)!='\*'){

\*(board+i\*width+j)=number\_boom+'0';

}

}

}

//檢查現在生成的board

/\*for(int i=0;i<height;i++){

for(int j=0; j<width;j++){

cout << \*(board+i\*width+j)<<" ";

}

cout << endl;

}\*/

}

void generate\_hint(char \*board, char\*play\_board,int height, int width){

int hints;

hints=round((sqrt(height\*width)));

cout << endl<<"hint :"<< hints << endl;

for(int i=0;i<hints;i++){

int x = rand()%height;

int y = rand()%height;

//cout << x << y <<endl;

while(\*(board+x\*width+y)=='\*'){

x = rand()%height;

y = rand()%height;

}

\*(play\_board+x\*width+y)=\*(board+x\*width+y);

}

//檢查現在生成的board

cout << "Play Board :"<< endl;

for(int i=0;i<height;i++){

for(int j=0; j<width;j++){

cout << \*(play\_board+i\*width+j)<<" ";

}

cout << endl;

}

}

int main(){

srand( time(NULL) );

int h,w;

cout << "請輸入地雷模板要多大 : (H\*W)";

cin >> h >> w;

cout << endl;

//生成N\*N的地雷棋盤

char game\_board[h][w];

generate\_board((char \*)game\_board,h,w);

cout << endl;

cout << "Game Board :"<< endl;

//檢查是否有將生成的board傳回來

for(int i=0;i<h;i++){

for(int j=0; j<w;j++){

cout << game\_board[i][j]<<" ";

}

cout << endl;

}

char play\_board[h][w];

for(int i=0;i<h;i++){

for(int j=0; j<w;j++){

play\_board[i][j]='-';

}

}

cout << endl;

//全空的play\_board

/\*cout << "Play Board :"<< endl;

//檢查play\_board

for(int i=0;i<N;i++){

for(int j=0; j<N;j++){

cout << play\_board[i][j]<<" ";

}

cout << endl;

}\*/

//自動產生幾個提示(round(sqrt(height))

generate\_hint((char \*)game\_board,(char \*)play\_board,h,w);

//開始準備玩踩地雷 boom 總共有幾個地雷 mark\_boom已標記的地雷

int booms=10;

cout << endl<<"遊戲開始 "<< endl <<endl;

//先把上面的輸入N抓掉

//cin.getline(sentence, 10);

//讀取執行的指令

int end=0;

//把周圍地雷是0的都打開

int loop=0;

while(end==0){

/\*cin.getline(sentence, 10);

if(sentence[0]=='o' && sentence[1]==' '){

cout << "open" <<endl;

cout << sentence[2] <<" "<< sentence[4] <<endl;

if(game\_board[sentence[2]-'0'][sentence[4]-'0']=='\*'){

cout << "Boom! End Game" << endl << endl;

end=1;

}

else{

play\_board[sentence[2]-'0'][sentence[4]-'0'] = game\_board[sentence[2]-'0'][sentence[4]-'0'];

}

}

else if(sentence[0]=='m' && sentence[1]==' '){

cout << "mark" <<endl;

cout << sentence[2] <<" "<< sentence[4] <<endl;

if(game\_board[sentence[2]-'0'][sentence[4]-'0']=='\*'){

play\_board[sentence[2]-'0'][sentence[4]-'0']='!';

mark\_boom++;

}

else{

play\_board[sentence[2]-'0'][sentence[4]-'0']='!';

}

}

else{

cout << endl << "error input 格式錯誤 ( 打開地雷 :o x y 標記地雷 :m x y )"<<endl<<endl;

}\*/

//把0和周圍沒有地雷的地方都直接打開

for(int i=0;i<h;i++){

for(int j=0;j<w;j++){

if(game\_board[i][j]=='0'){

play\_board[i][j]='0';

if(i>0 && j>0){

play\_board[i-1][j-1]=game\_board[i-1][j-1];

}

if(i>0){

play\_board[i-1][j]=game\_board[i-1][j];

}

if(i>0 && j+1<w){

play\_board[i-1][j+1]=game\_board[i-1][j+1];

}

if(j>0){

play\_board[i][j-1]=game\_board[i][j-1];

}

if(j+1<w){

play\_board[i][j+1]=game\_board[i][j+1];

}

if(i+1<h && j>0){

play\_board[i+1][j-1]=game\_board[i+1][j-1];

}

if(i+1<h){

play\_board[i+1][j]=game\_board[i+1][j];

}

if(i+1<h && j+1<w){

play\_board[i+1][j+1]=game\_board[i+1][j+1];

}

}

}

}

//把能開得先打開

for(int i=0;i<h;i++){

for(int j=0;j<w;j++){

//判斷是不是數字

if(play\_board[i][j]!='0' && play\_board[i][j]!='!' && play\_board[i][j]!='-' && play\_board[i][j]!='\*'){

int number=play\_board[i][j]-'0';

//一個一個檢查 假如有數字和周圍沒打開的格子數量一樣，則代表全部是炸彈

int space\_number=0;

if(i>0 && j>0 ){

if(play\_board[i-1][j-1]=='-'||play\_board[i-1][j-1]=='\*'){

space\_number++;

}

}

if(i>0){

if(play\_board[i-1][j]=='-'||play\_board[i-1][j]=='\*'){

space\_number++;

}

}

if( i>0 && j+1<w){

if(play\_board[i-1][j+1]=='-'||play\_board[i-1][j+1]=='\*'){

space\_number++;

}

}

if( j>0 ){

if(play\_board[i][j-1]=='-'||play\_board[i][j-1]=='\*'){

space\_number++;

}

}

if(j+1<w){

if(play\_board[i][j+1]=='-'||play\_board[i][j+1]=='\*'){

space\_number++;

}

}

if( i+1<h && j>0){

if(play\_board[i+1][j-1]=='-'||play\_board[i+1][j-1]=='\*'){

space\_number++;

}

}

if( i+1<h){

if(play\_board[i+1][j]=='-'||play\_board[i+1][j]=='\*'){

space\_number++;

}

}

if( i+1<h && j+1<w){

if(play\_board[i+1][j+1]=='-'||play\_board[i+1][j+1]=='\*'){

space\_number++;

}

}

if(space\_number==play\_board[i][j]-'0'){

play\_board[i-1][j-1]=game\_board[i-1][j-1];

play\_board[i-1][j]=game\_board[i-1][j];

play\_board[i-1][j+1]=game\_board[i-1][j+1];

play\_board[i][j-1]=game\_board[i][j-1];

play\_board[i][j+1]=game\_board[i][j+1];

play\_board[i+1][j-1]=game\_board[i+1][j-1];

play\_board[i+1][j]=game\_board[i+1][j];

play\_board[i+1][j+1]=game\_board[i+1][j+1];

}

//cout << "position " << " x : "<<i<<" y : " << j << "Boom : " << space\_number <<endl;

}

}

}

for(int i=0;i<h;i++){

for(int j=0;j<w;j++){

//判斷是不是數字

if(play\_board[i][j]!='0' && play\_board[i][j]!='!' && play\_board[i][j]!='-'){

int number=play\_board[i][j]-'0';

//假如數字=周圍已mark的炸彈數量，代表剩下的空格都是安全的

int boom\_number=0;

if( i>0 && j>0 && play\_board[i-1][j-1]=='\*'){

boom\_number++;

}

if( i>0 && play\_board[i-1][j]=='\*'){

boom\_number++;

}

if( i>0 && j+1<w && play\_board[i-1][j+1]=='\*'){

boom\_number++;

}

if( j>0 && play\_board[i][j-1]=='\*'){

boom\_number++;

}

if( j+1<w && play\_board[i][j+1]=='\*'){

boom\_number++;

}

if( i+1<h && j>0 && play\_board[i+1][j-1]=='\*'){

boom\_number++;

}

if( i+1<h && play\_board[i+1][j]=='\*'){

boom\_number++;

}

if( i+1<h && j+1<w && play\_board[i+1][j+1]=='\*'){

boom\_number++;

}

if(boom\_number==play\_board[i][j]-'0'){

play\_board[i-1][j-1]=game\_board[i-1][j-1];

play\_board[i-1][j]=game\_board[i-1][j];

play\_board[i-1][j+1]=game\_board[i-1][j+1];

play\_board[i][j-1]=game\_board[i][j-1];

play\_board[i][j+1]=game\_board[i][j+1];

play\_board[i+1][j-1]=game\_board[i+1][j-1];

play\_board[i+1][j]=game\_board[i+1][j];

play\_board[i+1][j+1]=game\_board[i+1][j+1];

}

}

}

}

//如果找到的炸彈和實際有的炸彈一樣多則結束 假如還沒結束 印出現在的板子

int found\_boom=0;

for(int i=0;i<h;i++){

for(int j=0;j<w;j++){

if(play\_board[i][j]=='\*'){

found\_boom++;

}

}

}

loop++;

if(loop>100){

cout << endl << "Loop" <<endl;

end=1;

}

if(found\_boom==booms){

end=1;

}

if(end==1){

for(int i=0;i<h;i++){

for(int j=0;j<w;j++){

cout << play\_board[i][j]<<" ";

}

cout <<endl;

}

}

}

cout << endl <<"GAME OVER"<<endl<<endl;

return 0;

}