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。

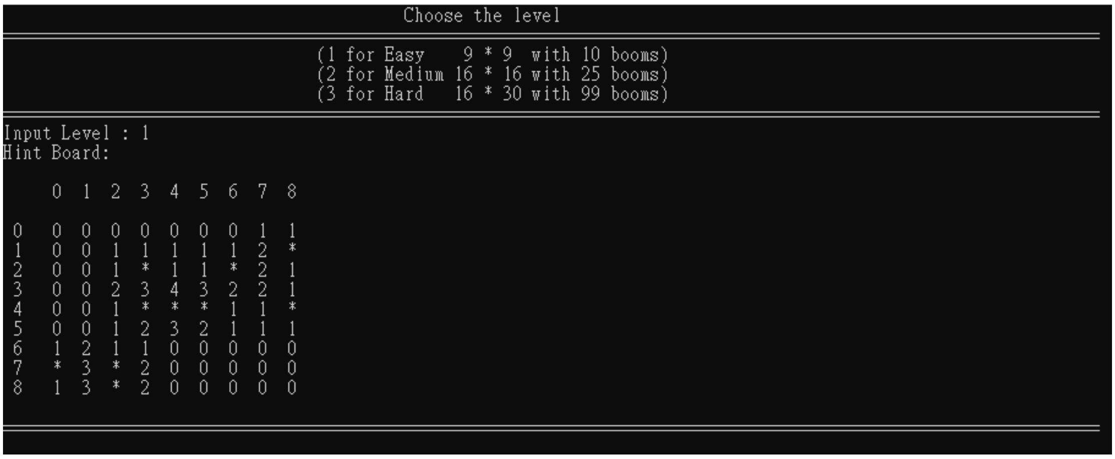
實作:

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

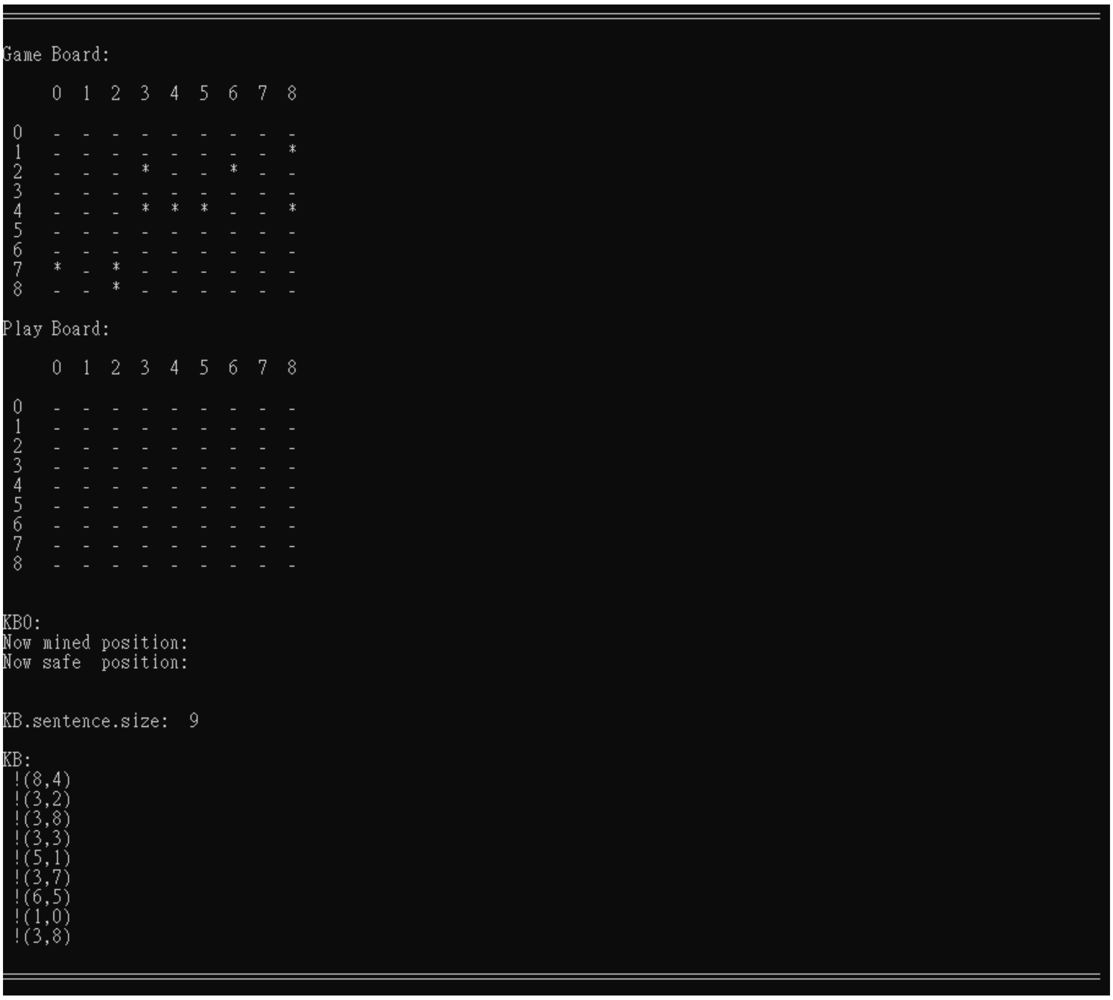
程式；

一開始先印出正確解答



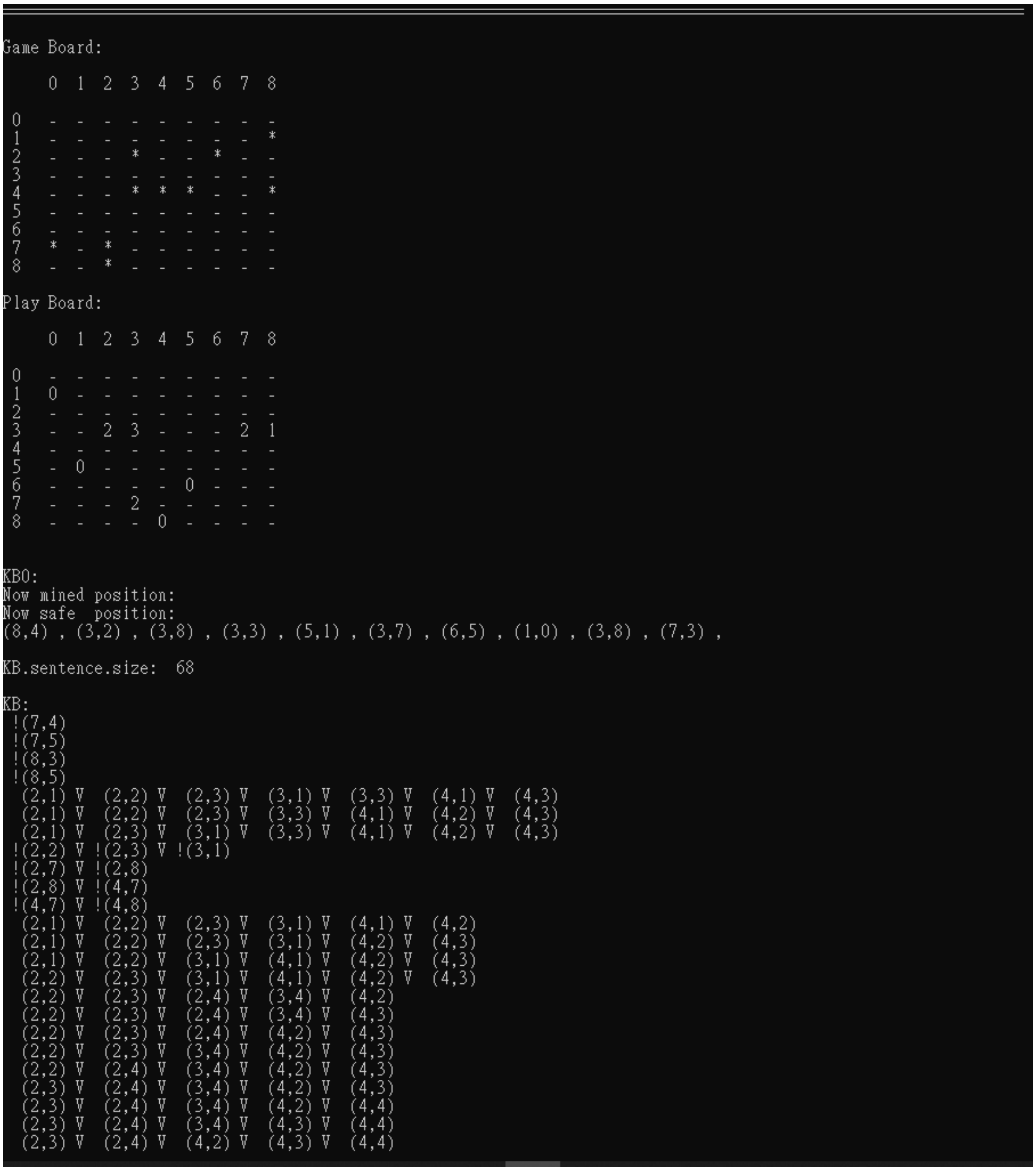
```
                                                    Choose the level
========================================================================================================
                                        (1 for Easy    9 * 9  with 10 booms)
                                        (2 for Medium 16 * 16 with 25 booms)
                                        (3 for Hard   16 * 30 with 99 booms)
========================================================================================================
Input Level : 1
Hint Board:

      0  1  2  3  4  5  6  7  8

 0    0  0  0  0  0  0  0  1  1
 1    0  0  1  1  1  1  1  2  *
 2    0  0  1  *  1  1  *  2  1
 3    0  0  2  3  4  3  2  2  1
 4    0  0  1  *  *  *  1  1  *
 5    0  0  1  2  3  2  1  1  1
 6    1  2  1  1  0  0  0  0  0
 7    *  3  *  2  0  0  0  0  0
 8    1  3  *  2  0  0  0  0  0

========================================================================================================
```

接下來是電腦看到的 BOARD 和玩家看到 BOARD 和最初的提示

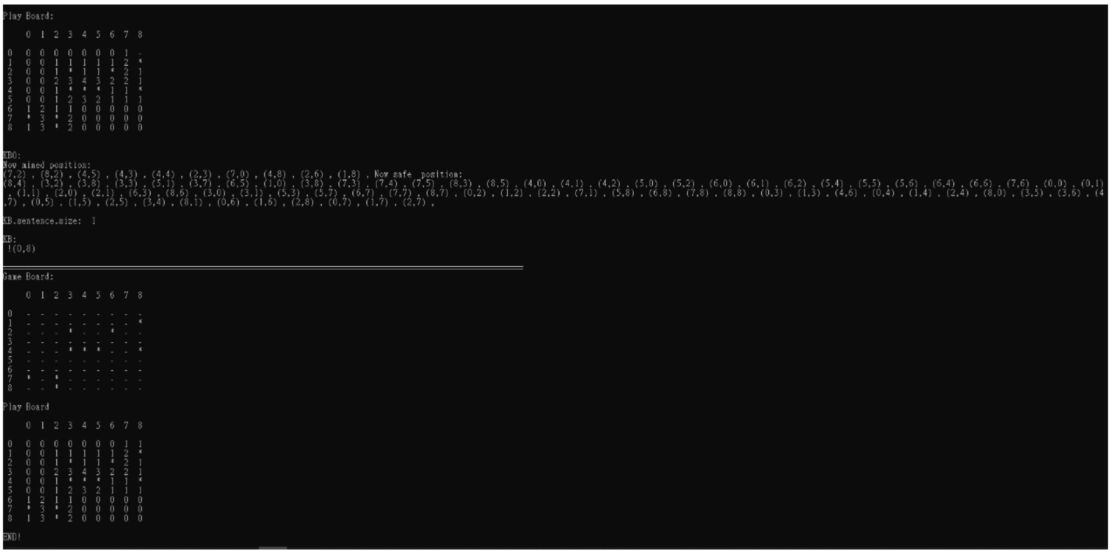


```
========================================================================================================

Game Board:

      0  1  2  3  4  5  6  7  8

 0    -  -  -  -  -  -  -  -  -
 1    -  -  -  -  -  -  -  -  *
 2    -  -  -  *  -  -  *  -  -
 3    -  -  -  -  -  -  -  -  -
 4    -  -  -  *  *  *  -  -  *
 5    -  -  -  -  -  -  -  -  -
 6    -  -  -  -  -  -  -  -  -
 7    *  -  *  -  -  -  -  -  -
 8    -  -  *  -  -  -  -  -  -

Play Board:

      0  1  2  3  4  5  6  7  8

 0    -  -  -  -  -  -  -  -  -
 1    -  -  -  -  -  -  -  -  -
 2    -  -  -  -  -  -  -  -  -
 3    -  -  -  -  -  -  -  -  -
 4    -  -  -  -  -  -  -  -  -
 5    -  -  -  -  -  -  -  -  -
 6    -  -  -  -  -  -  -  -  -
 7    -  -  -  -  -  -  -  -  -
 8    -  -  -  -  -  -  -  -  -


KB0:
Now mined position:
Now safe  position:


KB.sentence.size:  9

KB:
 !(8,4)
 !(3,2)
 !(3,8)
 !(3,3)
 !(5,1)
 !(3,7)
 !(6,5)
 !(1,0)
 !(3,8)

========================================================================================================
```

持續運算 KnowledgeBase， KnowledgeBase0

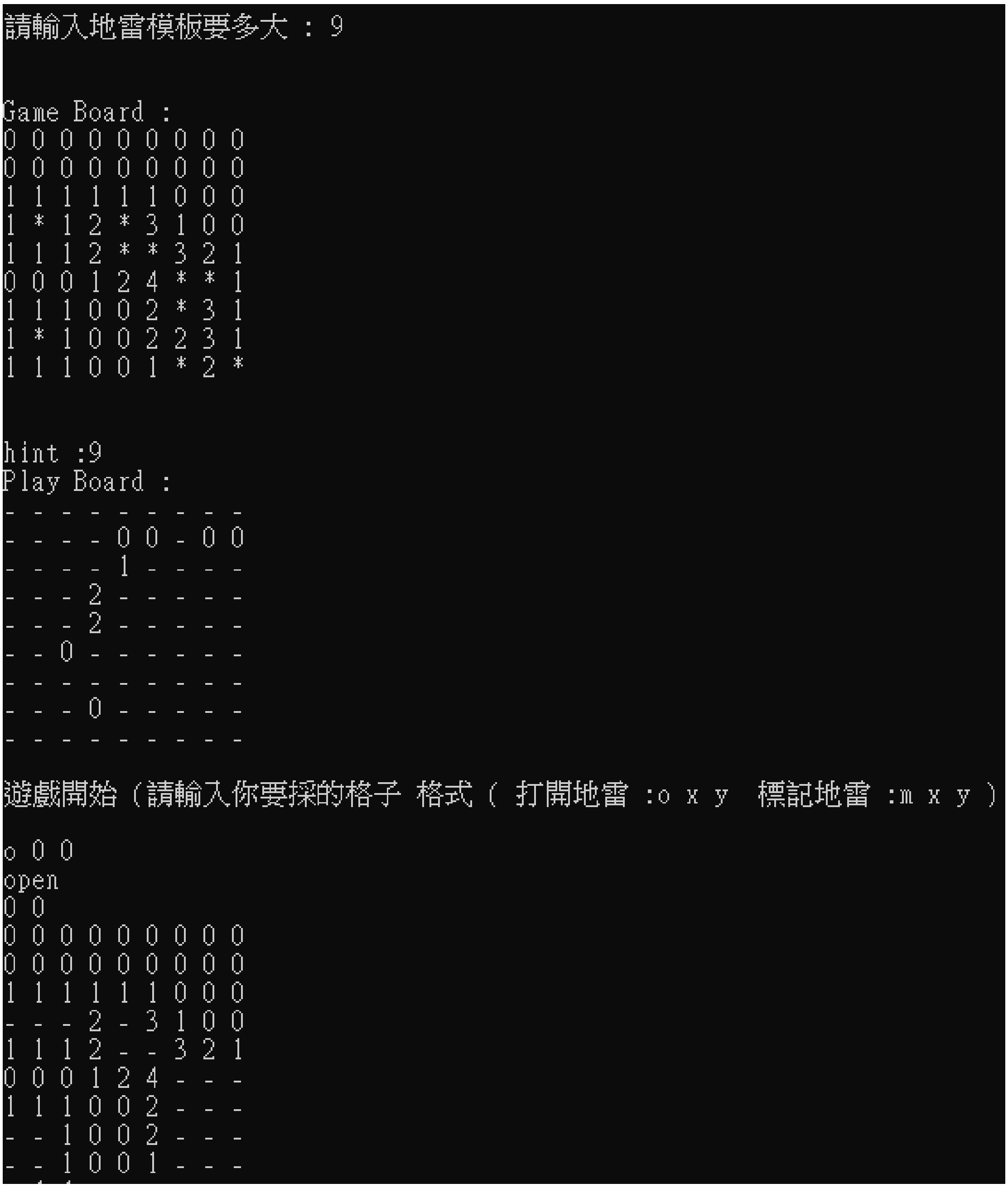


```
Game Board:

    0  1  2  3  4  5  6  7  8

 0  -  -  -  -  -  -  -  -  -
 1  -  -  -  -  -  -  -  -  *
 2  -  -  -  *  -  -  *  -  -
 3  -  -  -  -  -  -  -  -  -
 4  -  -  -  *  *  *  -  -  *
 5  -  -  -  -  -  -  -  -  -
 6  -  -  -  -  -  -  -  -  -
 7  *  -  *  -  -  -  -  -  -
 8  -  -  *  -  -  -  -  -  -

Play Board:

    0  1  2  3  4  5  6  7  8

 0  -  -  -  -  -  -  -  -  -
 1  0  -  -  -  -  -  -  -  -
 2  -  -  -  -  -  -  -  -  -
 3  -  -  2  3  -  -  -  2  1
 4  -  -  -  -  -  -  -  -  -
 5  -  0  -  -  -  -  -  -  -
 6  -  -  -  -  -  0  -  -  -
 7  -  -  -  2  -  -  -  -  -
 8  -  -  -  -  0  -  -  -  -


KB0:
Now mined position:
Now safe  position:
(8,4) , (3,2) , (3,8) , (3,3) , (5,1) , (3,7) , (6,5) , (1,0) , (3,8) , (7,3) ,

KB.sentence.size:  68

KB:
 !(7,4)
 !(7,5)
 !(8,3)
 !(8,5)
  (2,1) V  (2,2) V  (2,3) V  (3,1) V  (3,3) V  (4,1) V  (4,3)
  (2,1) V  (2,2) V  (2,3) V  (3,3) V  (4,1) V  (4,2) V  (4,3)
  (2,1) V  (2,3) V  (3,1) V  (3,3) V  (4,1) V  (4,2) V  (4,3)
 !(2,2) V !(2,3) V !(3,1)
 !(2,7) V !(2,8)
 !(2,8) V !(4,7)
 !(4,7) V !(4,8)
  (2,1) V  (2,2) V  (2,3) V  (3,1) V  (4,1) V  (4,2)
  (2,1) V  (2,2) V  (2,3) V  (3,1) V  (4,2) V  (4,3)
  (2,1) V  (2,2) V  (3,1) V  (4,1) V  (4,2) V  (4,3)
  (2,2) V  (2,3) V  (3,1) V  (4,1) V  (4,2) V  (4,3)
  (2,2) V  (2,3) V  (2,4) V  (3,4) V  (4,2)
  (2,2) V  (2,3) V  (2,4) V  (3,4) V  (4,3)
  (2,2) V  (2,3) V  (2,4) V  (4,2) V  (4,3)
  (2,2) V  (2,3) V  (3,4) V  (4,2) V  (4,3)
  (2,2) V  (2,4) V  (3,4) V  (4,2) V  (4,3)
  (2,3) V  (2,4) V  (3,4) V  (4,2) V  (4,3)
  (2,3) V  (2,4) V  (3,4) V  (4,2) V  (4,4)
  (2,3) V  (2,4) V  (3,4) V  (4,3) V  (4,4)
  (2,3) V  (2,4) V  (4,2) V  (4,3) V  (4,4)
```

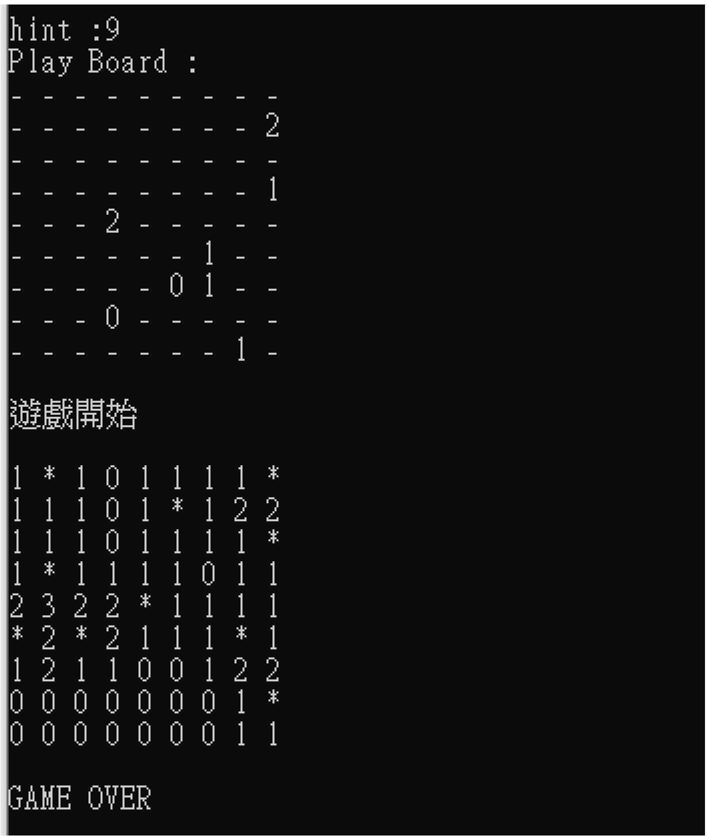


```
Play Board:

    0  1  2  3  4  5  6  7  8

 0  0  0  0  0  0  0  0  1  -
 1  0  0  1  1  1  1  1  2  *
 2  0  0  1  *  1  1  *  2  1
 3  0  0  2  3  4  3  2  2  1
 4  0  0  1  *  *  *  1  1  *
 5  0  0  1  2  3  2  1  1  1
 6  1  2  1  1  0  0  0  0  0
 7  *  3  *  2  0  0  0  0  0
 8  1  3  *  2  0  0  0  0  0


KB0:
Now mined position:
(7,2) , (8,2) , (4,5) , (4,3) , (4,4) , (2,3) , (7,0) , (4,8) , (2,6) , (1,8) , Now safe  position:
(8,4) , (3,2) , (3,8) , (3,3) , (5,1) , (3,7) , (6,5) , (1,0) , (3,8) , (7,3) , (7,4) , (7,5) , (8,3) , (8,5) , (4,0) , (4,1) , (4,2) , (5,0) , (5,2) , (6,0) , (6,1) , (6,2) , (5,4) , (5,5) , (5,6) , (6,4) , (6,6) , (7,6) , (0,0) , (0,1)
 , (1,1) , (2,0) , (2,1) , (6,3) , (8,6) , (3,0) , (3,1) , (5,3) , (5,7) , (6,7) , (7,7) , (8,7) , (0,2) , (1,2) , (2,2) , (7,1) , (5,8) , (6,8) , (7,8) , (8,8) , (0,3) , (1,3) , (4,6) , (0,4) , (1,4) , (2,4) , (8,0) , (3,5) , (3,6) , (4
,7) , (0,5) , (1,5) , (2,5) , (3,4) , (8,1) , (0,6) , (1,6) , (2,8) , (0,7) , (1,7) , (2,7) ,

KB.sentence.size:  1

KB:
 !(0,8)
================================================================
Game Board:

    0  1  2  3  4  5  6  7  8

 0  -  -  -  -  -  -  -  -  -
 1  -  -  -  -  -  -  -  -  *
 2  -  -  -  *  -  -  *  -  -
 3  -  -  -  -  -  -  -  -  -
 4  -  -  -  *  *  *  -  -  *
 5  -  -  -  -  -  -  -  -  -
 6  -  -  -  -  -  -  -  -  -
 7  *  -  *  -  -  -  -  -  -
 8  -  -  *  -  -  -  -  -  -

Play Board

    0  1  2  3  4  5  6  7  8

 0  0  0  0  0  0  0  0  1  1
 1  0  0  1  1  1  1  1  2  *
 2  0  0  1  *  1  1  *  2  1
 3  0  0  2  3  4  3  2  2  1
 4  0  0  1  *  *  *  1  1  *
 5  0  0  1  2  3  2  1  1  1
 6  1  2  1  1  0  0  0  0  0
 7  *  3  *  2  0  0  0  0  0
 8  1  3  *  2  0  0  0  0  0

END!
```

這是驗算用的



```
請輸入地雷模板要多大 : 9


Game Board :
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0
1 * 1 2 * 3 1 0 0
1 1 1 2 * * 3 2 1
0 0 0 1 2 4 * * 1
1 1 1 0 0 2 * 3 1
1 * 1 0 0 2 2 3 1
1 1 1 0 0 1 * 2 *


hint :9
Play Board :
- - - - - - - - -
- - - - 0 0 - 0 0
- - - - 1 - - - -
- - - 2 - - - - -
- - - 2 - - - - -
- - 0 - - - - - -
- - - - - - - - -
- - - 0 - - - - -
- - - - - - - - -

遊戲開始 (請輸入你要採的格子 格式 ( 打開地雷 :o x y  標記地雷 :m x y )

o 0 0
open
0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0
- - - 2 - 3 1 0 0
1 1 1 2 - - 3 2 1
0 0 0 1 2 4 - - -
1 1 1 0 0 2 - - -
- - 1 0 0 2 - - -
- - 1 0 0 1 - - -
```



```
hint :9
Play Board :
- - - - - - - - -
- - - - - - - - 2
- - - - - - - - -
- - - - - - - - 1
- - - 2 - - - - -
- - - - - - 1 - -
- - - - - 0 1 - -
- - - 0 - - - - -
- - - - - - - 1 -

遊戲開始

1 * 1 0 1 1 1 1 *
1 1 1 0 1 * 1 2 2
1 1 1 0 1 1 1 1 *
1 * 1 1 1 1 0 1 1
2 3 2 2 * 1 1 1 1
* 2 * 2 1 1 1 * 1
1 2 1 1 0 0 1 2 2
0 0 0 0 0 0 0 1 *
0 0 0 0 0 0 0 1 1

GAME OVER
```

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++;
都是安全的
                        }
                    }
                    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;
}
```