實作方法：

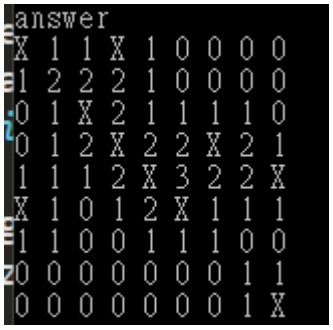
用 KB 來存 single-literal clause，如果 single-literal clause 就是優先處理，看它是地雷還是 hint，如果是地雷就標記它，如果不是就點開它，看它的 hint 是多少，再用它的 hint 標記 clause，如果 clause 長度小於或和另一個 clause 一樣就不執行，再 push 回 KB。如果都沒有 single-literal clause 就做 pairwise matching，就產生新的 clause，如果連續五回合(我設定的)沒有產生新的 clause，它就卡住了。matchmatch 會檢查兩個 clause 是否有一樣的內容，如果有一樣的內容就會刪掉其中一個。build 會處理 C 幾取幾的問題，把原本的 clause 改成 spec 上的樣子。dealing 會從 KB 裡面看有沒有可以用的 clause (single clause)，如果確定某個 clause 是確定的值就把它丟進 KB0，如果是地雷就 mark 它，如果是 hint 就點它，然後得到新的 clause。creategraph 會先建立邊界，然後取 random 的 i 和 j 來製作 graph。

分析結果:

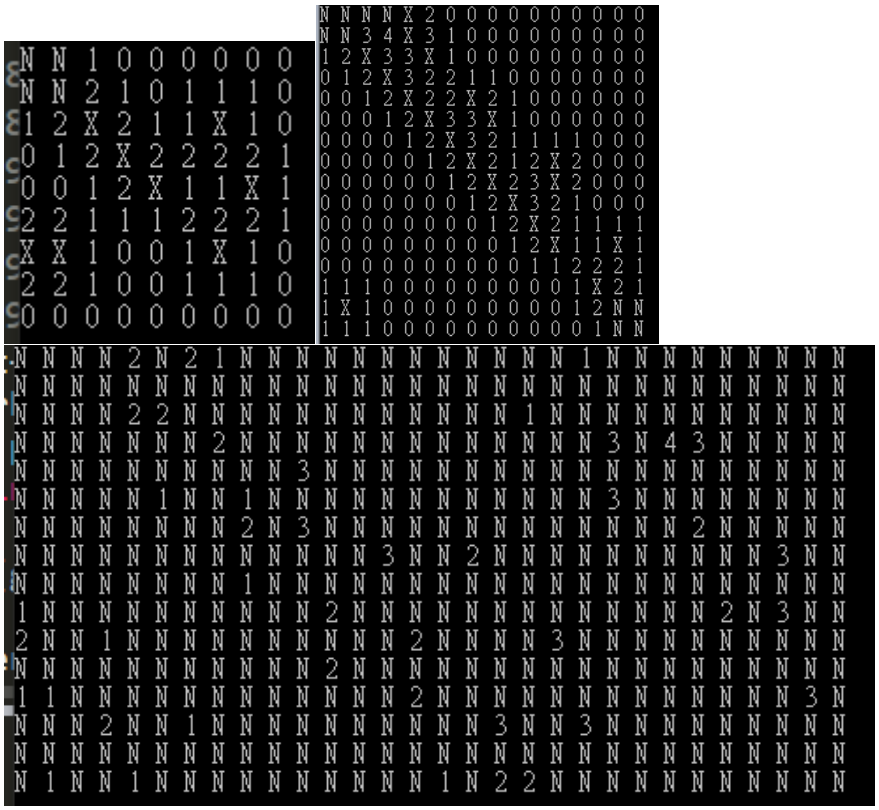
十五次中成功解開的百分比例

Safe 值 / 難度	**簡單**	**中等**	**困難**
round(sqrt(width * height))	**0.53**	**0.4**	**0.2**
round((width * height) / 3)	**0. 46**	**0.2**	**0.07**
round(sqrt(width * height) *2)	**0.53**	**0.13**	**0.07**

成功的圖片：



失敗的照片：



心得：要搞清楚 and 和 or 的那些 clause 就花了很多時間，還查了很多資料，用人腦算感覺還沒有到非常困難，但要再轉成用程式就耗費了很多心力，真的覺得很難，尤其是判斷 clause 有沒有重複和可以消除掉的內容，想了很多要如何實作。從資料的結構我也很不熟練，不太會用 pair，特別是雙層的 pair 更是常常搞錯 first 跟 second 現在存了什麼值。每次跑踩地雷的時間也要很久，心很累。

Observation：從跑的結果來看，hint 如果比較密集好像比較容易成功解開，然後愈接近邊緣的地方如果卡住，常常就會一整片都解不開，沒有結果。發現 safe 值愈大，成功解開的機率愈高。

[Optional / Extra Credits] These are for discussion only; no implementation/experiments required.

How to use first-order logic here?

first-order logic 可以表現出相鄰的值有什麼關係

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

Backward chaining比Forward chaining花費的時間可能會比forward chaining少，因為它是有目標的，forward chaining是隨機挑選的。

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

在快要stuck的時候檢查有沒有可以融合在一起的clause，或是從比較短的clause的開始處理

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

Assignment#2 有用到 heuristic，用這個方式可以按照順序的優先來排序，可能可以提高解開踩地雷的成功機率。

```
#include <bits/stdc++.h>

using namespace std;

//-1 => mine   -10 => bound
int graph[35][35];
//-1 => mark as mine     -2 => unknown       -10 => bound
int graphsolution[35][35];
int x[8] = {1, 1, 1, -1, -1, -1, 0, 0};
int y[8] = {0, 1, -1, 0, 1, -1, 1, -1};
int unfound, unmarked;
std::vector<vector<pair<bool,pair<int, int>>>> kobe;
std::vector<pair<bool, pair<int, int>>> kobe0;


//if true then push, if false then don't push
bool pushin_kobe(std::vector<pair<bool, pair<int, int>>> v){
    for(pair<bool, pair<int, int>> &f : kobe0){
        int i = 0;
        while(i < v.size()){
            if(v[i].second.first == f.second.first && v[i].second.second ==
f.second.second){
```

```
                        if(v[i].first == f.first){
                                return false;
                        }
                        else{
                                v.erase(v.begin() + i);
                                i -= 1;
                        }
                        break;
                }
                i++;
        }
    }
    int y = 0;
    while(y < kobe.size()){
        int samecnt = 0;
        for(pair<bool, pair<int, int>> &t : kobe[y]){
            for(pair<bool, pair<int, int>> &f : v){
                if(f.first == t.first && f.second.second == t.second.second &&
f.second.first == t.second.first){
                        samecnt ++;
                        break;
                }
            }
        }
        if(v.size() == samecnt){
            //check if kobe clause <= v
            if(kobe[y].size() <= v.size()){
                return false;
            }
            else{
                kobe.erase(kobe.begin() + y);
                y--;
            }

        }
        // check if kobe's size is equal to v's size
        else if(kobe[y].size() == samecnt){
            return false;
        }
        y++;
    }
    kobe.push_back(v);
    return true;
}

int matchmatch(std::vector<pair<bool, pair<int, int>>> &s, std::vector<pair<bool,
pair<int, int>>> &t){
```

```
        int comp = 0;
        int index = 0;
        for(pair<bool, pair<int, int>> &ss : s){
                for(pair<bool, pair<int, int>> &tt : t){
                        if(tt.second.second == ss.second.second && ss.second.first ==
tt.second.first){
                                if(ss.first != tt.first){
                                        comp += 1;
                                        break;
                                }
                                else{
                                        index += 1;
                                }
                        }
                }
        }
        //keeping s's value
        if(s.size() == index)  return 1;
        //keeping t's value
        else if(t.size() == index)   return 2;
        else if(comp == 1){
                vector<pair<bool, pair<int, int>>> r;
                vector<pair<bool, pair<int, int>>> st = s;
                st.insert(st.end(), t.begin(), t.end());
                sort(st.begin(), st.end(),[](pair<bool, pair<int, int>> &a, pair<bool, pair<int,
int>> &b){
                        if(b.second.first == a.second.first){
                                return a.second.second < b.second.second;
                        }
                        return a.second.first < b.second.first;

                });
                int i = 1;
                while(i < st.size()){
                        if(st[i-1].second.first == st[i].second.first && st[i-1].second.second ==
st[i].second.second){
                                if(st[i-1].first != st[i].first){
                                        st.erase(st.begin() + i - 1, st.begin() + i + 1);
                                        i -= 2;
                                }
                                else{
                                        st.erase(st.begin() + i);
                                        i -= 1;
                                }
                        }
                        i += 1;
                }
```

```
            r = st;
            //make a new clause
            if(pushin_kobe(r)){
                return 3;
            }
        }
        return 0;
}


void build(std::vector<pair<bool, pair<int, int>>> &v, int mine) {
            // By default, input clause's first will be false
            if (mine == 0) {
                for (auto &f : v) {
                    pushin_kobe(std::vector<pair<bool, pair<int, int>>>{f});
                }
            }
            else if (mine == v.size()) {
                for (auto &f : v) {
                    f.first = true;
                    pushin_kobe(std::vector<pair<bool, pair<int, int>>>{f});
                }
            }
            else {
                // Generate C(m, n+1) clauses, each having n+1 negative literals
                std::vector<bool> per(v.size() - mine - 1, false);
                per.insert(per.end(), mine + 1, true);
                do {
                    std::vector<pair<bool, pair<int, int>>> condn;
                    for (size_t i = 0; i < per.size(); ++i) {
                        if (per[i]) {
                            condn.push_back(v[i]);
                        }
                    }

                    pushin_kobe(condn);
                } while (std::next_permutation(per.begin(), per.end()));


                // Generate C(m, m-n+1) clauses, each having m-n+1 positive literals
                for (auto &f : v) {
                    f.first = true;
                }

                per.clear();
                per.resize(mine - 1, false);
                per.insert(per.end(), v.size() - mine + 1, true);
```

```
                do {
                        std::vector<pair<bool, pair<int, int>>> condn;
                        for (size_t i = 0; i < per.size(); ++i) {
                                if (per[i]) {
                                        condn.push_back(v[i]);
                                }
                        }

                        pushin_kobe(condn);
                } while (std::next_permutation(per.begin(), per.end()));
        }
}

void showgraph(int width, int height, int _sol){    // _sol = 1 -> graph, _sol = 0 ->
graphsolution, -1 -> anwser
        cout << '\n';
        if (_sol == 1)
        {
                for (int i = 1; i < height + 1; ++i)
                {
                        for (int j = 1; j < width + 1; ++j)
                        {
                                //cout << "ckeck2\n";
                                if ( graph[i][j] == -1)
                                {
                                        cout << 'X' << ' ';
                                }
                                else cout << graph[i][j] << ' ';
                        }
                        cout << '\n';
                }
        }
        else if( _sol == 0)
        {
                for (int i = 1; i < height + 1; ++i)
                {
                        for (int j = 1; j < width + 1; ++j)
                        {
                                //cout << "ckeck2\n";
                                if ( graphsolution[i][j] == -2)
                                {
                                        cout << 'N' << ' ';
                                }
                                else if ( graphsolution[i][j] == -1)
                                {
                                        cout << 'X' << ' ';
                                }
```

```
                else cout << graphsolution[i][j] << ' ';
            }
            cout << '\n';
        }
    }
    else if( _sol == -1){
        cout << "answer\n";
        for (int i = 1; i < height + 1; ++i)
        {
            for (int j = 1; j < width + 1; ++j)
            {
                //cout << "ckeck2\n";
                if ( graphsolution[i][j] == -1)
                {
                    cout << 'X' << ' ';
                }
                else if ( graphsolution[i][j] == -2)
                {
                    cout << graph[i][j] << ' ';
                }
                else cout << graphsolution[i][j] << ' ';
            }
            cout << '\n';
        }
    }
    cout << '\n';
}


void dealing(int mine, int width, int height, int a){
    int cant = 5;
    vector<pair<bool, pair<int, int>>> nowkobe;
    while( !kobe.empty()){
        //cout << cant << '\n';
        if (unfound == 0) break;
        int single = 0;
        int i = 0;
        while(i < kobe.size()){
            if(kobe[i].size() == 1){
                nowkobe = kobe[i];
                single = 1;
                kobe.erase(kobe.begin() + i);
                break;
            }
            i += 1;
        }
        //there is not only one single literial
```

```
        if(single == 0){
            //cout << "have two\n";
            std::vector<pair<bool, pair<int, int>>> cond;
            bool newcla = false;
            int i = 0;
            while(i < kobe.size() ){
                if(kobe[i].size()<3){
                    int j = i + 1;
                    while(j < kobe.size()){
                        if(kobe[j].size() < 3 && j != i){
                            int flag = matchmatch(kobe[i], kobe[j]);
                            if(flag == 1){
                                kobe.erase(kobe.begin() + j);
                                j -= 1;
                            }
                            else if(flag == 2){
                                kobe.erase(kobe.begin() + i);
                                i -= 1;
                            }
                            else if(flag == 3){
                                newcla = true;
                                cant = 5;
                            }
                        }
                        j += 1;
                    }
                }
                i += 1;
            }
            if( newcla == false){
                cant -= 1;
                if( cant == 0){
                    cout<<"I can't\n";
                    //show the graph
                    // for (int i = 0; i < kobe.size(); ++i)
                    // {
                    //     for (int j = 0; j < kobe[i].size(); ++j)
                    //     {
                    //         cout << "bool: " << kobe[i][j].first << ", pair( " <<
kobe[i][j].second.first << ", " << kobe[i][j].second.second << ") / ";
                    //     }
                    //     cout << '\n';
                    // }
                    int i = 1, j = 1;
                    while(i <= height){
                        j = 1;
                        while(j <= width){
```

```
                        int num = graphsolution[i][j];
                        if(num == -1){
                            cout<< 'X' << ' ';
                        }
                        else if(num == -2){
                            cout << 'N' << ' ';
                        }
                        else{
                            cout<< num << ' ';
                        }
                        j += 1;
                    }
                    cout<<'\n';
                    i += 1;
                }
                cout<<'\n';
                return;
            }
            else if(cant == 1){
                for(pair<bool, pair<int, int>> &a : kobe0){
                    std::vector<pair<bool, pair<int, int>>> cond{a};
                    int i = 0;
                    while(i < kobe.size()){
                        if(matchmatch(cond, kobe[i]) == 1){
                            kobe.erase(kobe.begin() + i);
                            cant = 5;
                            i -= 1;
                        }
                        i += 1;
                    }
                }
            }
        }
    }
    //there is only one single literial
    else{
        //cout << "onlu one\n";
        int i = 0;
        cant = 5;
        pair<bool, pair<int, int>> nowcla = nowkobe.front();
        kobe0.push_back(nowcla);
        //match
        while(i < kobe.size()){
            if(matchmatch(nowkobe, kobe[i])== 1){
                kobe.erase(kobe.begin() + i);
                i -= 1;
            }
```

```
                i += 1;
            }

            //if nowcla.first is true => mine, is false => hint
            if(nowcla.first == false){
                int graphsol =
graphsolution[nowcla.second.first][nowcla.second.second];
                int graphcla =
graph[nowcla.second.first][nowcla.second.second];
                if(graphcla == -1){
                    cout<<"You touched mine!!\n";    //gameover
                    return;
                }
                else if(graphsol != -10){
                    if(graphsol == -2){
                        unmarked -= 1;

    graphsolution[nowcla.second.first][nowcla.second.second] = graphcla;
                    }
                    vector<pair<bool, pair<int, int>>> cond;
                    int j = 0;
                    //int mine = graphsol;
                    while( j < 8){
                        int num = graphsolution[nowcla.second.first +
x[j]][nowcla.second.second +y[j]];
                        if(num == -2){  //haven't been marked
                            pair<int, int> tmp = make_pair(nowcla.second.first
+ x[j], nowcla.second.second + y[j]);
                            cond.push_back(make_pair(false, tmp));
                        }
                        else if(num == -1){
                            graphcla -= 1;
                        }
                        j++;
                    }
                    //cout << "(" << nowcla.second.first << ", " <<
nowcla.second.second << ") =" << cond.size() << ", " << mine << '\n';
                    build(cond, graphcla);
                }
            }
            else if(nowcla.first == true){
                graphsolution[nowcla.second.first][nowcla.second.second] = -1;
                unfound -= 1;
            }
            if( unfound == 5 && a >= 1){
                std::vector<pair<bool, pair<int, int>>> cond;
                int i = 1, j = 1;
```

```
                        while(i <= height){
                                j = 1;
                                while(j <= width){
                                        if(graphsolution[i][j] == -2){
                                                pair<int, int> tmp = make_pair(i, j);
                                                cond.push_back(make_pair(false, tmp));
                                        }
                                        j++;
                                }
                                i++;
                        }
                        build(cond, unfound);
                }
        }
    }
    //show the graph
    cout<<"answer\n";
    int i = 1, j = 1;
    while(i <= height){
        j=1;
        while(j <= width){
            int num = graphsolution[i][j];
            if(num == -1){
                cout<< 'X' << ' ';
            }
            else if(num == -2){
                cout<< graph[i][j] << ' ';
            }
            else{
                cout<< num << ' ';
            }

            j += 1;
        }
        cout<< '\n';
        i += 1;
    }
    cout<<'\n';
    //cout << kobe0.size() << '\n';
}



void creategraph(int mine, int width, int height){
    int i = 0, j = 0;
    while(i <= height + 1){
        graph[i][0] = -10;
```

```
        graphsolution[i][0] = -10;
        graph[i][width+1] = -10;
        graphsolution[i][width+1] = -10;
        i += 1;
}
while(j <= width + 1){
        graph[0][j] = -10;
        graphsolution[0][j] = -10;
        graph[height+1][j] = -10;
        graphsolution[height+1][j] = -10;
        j += 1;
}
while(mine > 0){
        srand(time(NULL));
        i = rand() % height + 1;
        srand(time(NULL));
        j = rand() % width + 1;
        if(graph[i][j] == -1){
                continue;
                //cout<< i <<' ' << j << '\n';
        }
        mine -= 1;
        graph[i][j] = -1;
        int k = 0;
        while(k < 8){
                int tmp = graph[i + x[k]][j + y[k]];
                if(tmp >= 0){
                        graph[i + x[k]][j + y[k]] += 1;
                }
                k += 1;
        }

}
i = 1, j = 1;
cout<<'\n';
while(i <= height){
        j = 1;
        while(j <= width){
                int tmp = graph[i][j];
                if(tmp == -1){
                        cout<< 'X' << ' ';
                }
                else{
                        cout<< tmp << ' ';
                }
                j += 1;
        }
```

```
            i += 1;
            cout<<'\n';
        }
        cout<<'\n';
}
void safecell(int mine, int width, int height){
        int count = 0, safe = round(sqrt(width * height));
        cout << safe << '\n';
        int tari, tarj;
        stack<pair<int, int>> getzero;
        while(count < safe){
            while(!getzero.empty()){
                getzero.pop();
            }
            srand(time(NULL));
            int i = ( rand() % height ) + 1;
            srand(time(NULL));
            int j = ( rand() % width ) + 1;
            if (graph[i][j] != 0 && graph[i][j] != -1 && graphsolution[i][j] == -2)
            {
                graphsolution[i][j] = graph[i][j];
                count++;
                unmarked--;
                kobe.push_back(std::vector<pair<bool, pair<int,
int>>>{make_pair(false, make_pair(i, j))});
                continue;
            }
            else if (graph[i][j] == 0)
            {
                getzero.push(make_pair(i, j));
            }
            while(!getzero.empty()){
                //cout << "check " << count << '\n';
                pair<int, int> now = getzero.top();
                getzero.pop();
                graphsolution[now.first][now.second] = 0;
                count++;
                unmarked--;
                kobe.push_back(std::vector<pair<bool,
pair<int,int>>>{make_pair(false, make_pair(now.first, now.second))});
                for (int k = 0; k < 8; ++k)
                {
                    int X = now.first + x[k];
                    int Y = now.second + y[k];
                    if (graphsolution[X][Y] == -2)
                    {
                        if (graph[X][Y] == 0)
```

```
                        {
                                getzero.push(make_pair(X, Y));
                        }
                        graphsolution[X][Y] = graph[X][Y];
                        count++;
                        unmarked--;
                        kobe.push_back(std::vector<pair<bool, pair<int,
int>>>{make_pair(false, make_pair(X, Y))});
                }
            }
        }
        cout << count << '\n';
        //cout << "end\n";
        cout << '\n';

        for (int i = 1; i < height + 1; ++i)
            {
                for (int j = 1; j < width + 1; ++j)
                {
                    //cout << "ckeck2\n";
                    if ( graphsolution[i][j] == -2)
                    {
                        cout << 'N' << ' ';
                    }
                    else if ( graphsolution[i][j] == -1)
                    {
                        cout << 'X' << ' ';
                    }
                    else cout << graphsolution[i][j] << ' ';
                }
                cout << '\n';
            }
}


int main(){
        int c;
        cin>>c;
        for(int i=0;i<35;i++){
            for(int j=0;j<35;j++){
                graph[i][j] = 0;
                graphsolution[i][j] = -2;
            }
        }
        int i = 0, j = 0;
```

```
        kobe.clear();
        kobe0.clear();
        if(c == 0){
                unfound = 10;
                unmarked = 9 * 9;
                creategraph(10, 9, 9);
                safecell(10, 9, 9);
                dealing(10, 9, 9, c);
        }
        else if(c == 1){
                unfound = 25;
                unmarked = 16 * 16;
                creategraph(25, 16, 16);
                safecell(25, 16, 16);
                dealing(25, 16, 16, c);
        }
        else if(c == 2){
                unfound = 99;
                unmarked = 30 * 16;
                creategraph(99, 30, 16);
                safecell(99, 30, 16);
                dealing(99, 30, 16, c);
        }
}
```