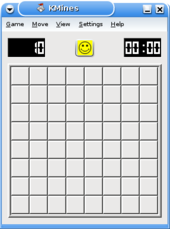
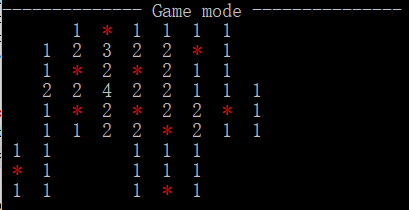
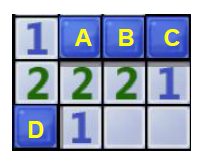
人工智慧概論

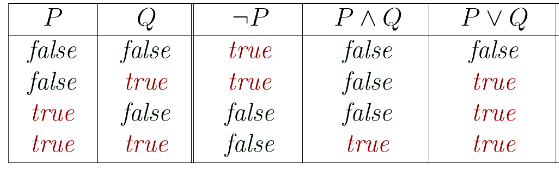
0613413 蔡怡君

－實驗目的：踩地雷，Minesweeper，生成一個MAP並使用Logic去找到結果，例如：

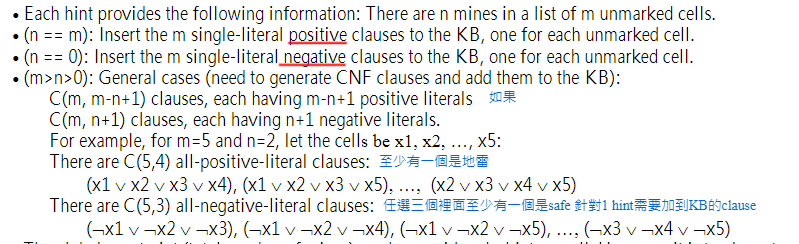
找到結果

－實驗方法：(The Rule of Resolution)  
－使用的概念：使用**Propositional Logic**，類似下圖這樣：

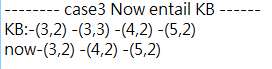
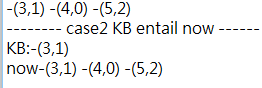
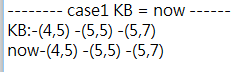


如何應用到地雷上面呢？請看右邊的圖：m=未填值，n=應該要填的mine數量。  
想法：假設一個hint 周圍有4個地雷，有8個neighbor，比如說已經有三個已經標記（2個地雷跟1個safe），這時n =2 m=5，安全的格子最多是3個就是m-n，所以說m-n+1=4個裡面至少有一個是地雷去生成Positive literals，而n+1個=3個至少有一個是safe所以可以去生成Negative literals。

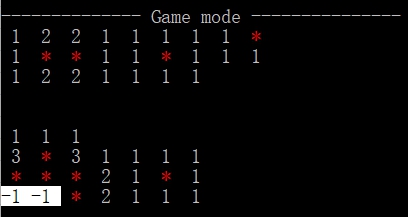
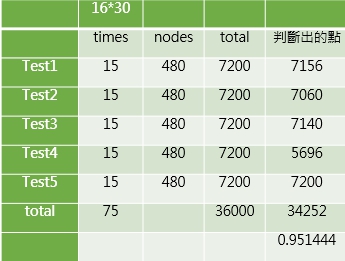
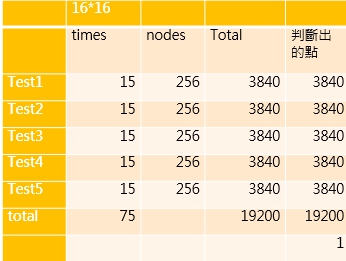
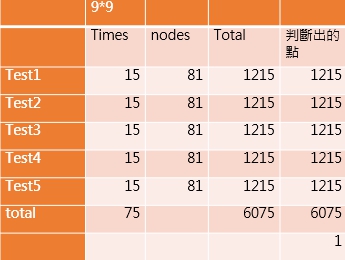
1. Case 1 ( m == n )：Insert m個single-literal postive，ex.看左上角的hint=1時，CNF為+A。
2. Case 2 ( n == 0 )：地雷都已經被給值了，所以剩下的都是-A、-B等等，在右圖的例子無法找到。
3. Case 3 (m > n > 0)：提供Combination of Positive、Negative。ex. A下面的hint = 2時，假設Case 1將A填值為true了，剩下1個mine，unmarked = 2，所以這時CNF會有+C(2,2)與-C(2,2)。

以此類推推出**公式**：

Control mode：生成地圖 提供一開始的Safe Cell，數量為地圖round(sqrt(R\*C))，ex. 9\*9地圖提供9個safe cell。  
KB0包含每一個cell對應的clause，KB則是需要判斷的clause。  
 Game mode：使用Logic clause去得到結果。  
While(還沒達到Game Termination){  
 if(KB裡頭有single-lateral clause)：  
 看clause是-還是+：-的話mark safe，+的話mark 地雷。  
 將此clause移至KB0  
 一對多Matching－使用這個判斷完的Cell將原KB裡的clause(含有此Cell判斷)去刪減：  
 刪減完的句子，要先去判斷有沒有same in KB or KB0、或是entail。(重要)  
 if 判斷完的Cell is Safe：  
 使用公式去生成CNF。  
 Otherwise：  
 多對多Matching－使用KB裡頭長度<=2的clauses  
 只有one pair of complementary literals可以去做resolution生成CNF。例如：– x2 v x3 與   
 x4 v x2 生成x3 v x4  
 生成的句子要去檢查有沒有duplication、Subsumption (只保留較Strict clause)  
 在KB or KB0裡頭c或是entail。  
 如果連續多對多Matching大於10次，而且沒有生出single-literal，那就代表Stuck →Break  
}  
Notice：Insert 進去之前都要檢查： ( 以下呈現為程式結果 )



－結果觀察：**Time對於Safe Cell的數量改變：(討論在後面)**

**正確度分析、完整度(Completeness): (是否stuck？)**以下表格為一次程式跑15次總共跑75次，而這15次其中包含又有safe cell的改變，Test資料為時間分析的資料繼續使用，可以看到在Eazy跟Medium裡頭幾乎都是百分之百的正確，但並不代表不會面對到Stuck的狀態，在解釋完正確度分析後會談論到此。   
****

圖四.9\*9也會遇到stuck  
(在其他次測試中發現)

－結果分析：

就**時間對於Safe Cell數量**的分析去看，一開始還沒有進行分析前，我以為提供更多的Safe Cell應該會使得時間跑得更快，但是事實證明，**提供更多的Safe Cell會讓時間更慢**因為曲線是向上的，只要有足夠的Safe Cell去保證說第一開始不會走到地雷或是足以展開，就可以得到解。我也有**嘗試將9\*9的safe cell調整至2、3**(原為9，實驗最低6)，而這邊**取決於機率會不會取點取到取到快速展開**，例如hint = 0，如果都取到有hint值例如2、3，這邊會很容易遇到stuck！所以Safe Cell還是要取適當的數量讓Time去optimize，**實驗的結果為取比Round(sqrt(R\*C))較小的數一點的數都可以更優化。**

就**正確度分析**來說，Easy、Medium在實驗的75次過程中都是不會遇到stuck的，但其實是**事實上是機率極低可以看圖四**，而在實驗中Hard則是十次裡頭只有一兩次可以完整解出，雖然正確率高，但完整度比較低，這應都是取決於一開始給得Safe Cell，**如果一開始給的Safe Cell很密集於某一區，這樣更容易遇到Stuck的狀況**，而Safe Cell越分散應越不容易得到Stuck，又

或是地雷分布密集。  
－Discussion：

**Question 1： How to use first-order logic here？**

**Answer1：**可以用以下的Logic ：  
Hint (P(y),n) 表示在點y的8個direction有n個Mine。

M(P(y)) 表示在點y是否有Mine。

Neighbor(y) 點y的鄰居。

y  M(P(Neighbor(y)) ⬄ Hint(P(y), 1)  如果點P(y) Hint =1代表存在1個鄰居是mine。

y  ~M(P(Neighbor(y)) ⬄ Hint(P(y)), 0) 如果點P(y) Hint =0代表所有鄰居都不是Mine。

使用FOL時，可以省去非常多Propositional logic的變數記憶體空間，同時也可以使得最後的結果較為簡潔，不用窮舉！

**Question 2： Discuss whether forward chaining or backward chaining applicable to this**

**problem？**

**Answer 2：**我認為在這裡使用Forward chaining，應該會比backward更為適合因為地雷問題，因為在地雷問題中，最一開始展開時除了我們已知的Safe Cell List以外不會擁有太多其餘的資訊，在這裡好像沒有辦法從Goal去出發，假如一開始是Hint = 2出發，去檢視它的horn clause，因為沒有其他的資訊，反而使用FC比較合理，而使用BC回推的話會較沒有幫助，而展開到一定的數量時再去使用BC會比較合理，也會花費比FC更少的時間，因為FC會比較偏向亂槍打鳥。

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

**Answer 3：猜測根據機率－**當遇到stuck的時候，這邊可以使用剩下有關這幾個Point的資訊去猜測，例如剩下的有A、B、C，分別去看這三個影響到的clause數量，去決定它的機率，例如：假設這張圖只剩下2顆地雷要放，而2下方的3右邊、右下都有地雷了，這時候去判斷說1=mine時2就會=safe，而4、5、6都有可能會是mine同時3=safe，而如果當2=mine時，1、4、5、6=safe同時3=mine，這時候可以去算機率1是地雷是3/4，而2是1/4，所以去猜測1為地雷。

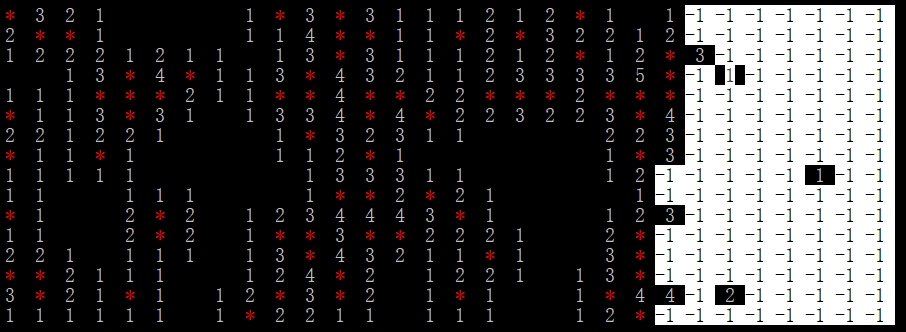
**Question 4：Discuss ideas of modifying the method in Assignment#2 to solve the current problem.  
Answer 4：**如果要將這次的作業套用Assignment2的話，應該要先展開一定的地圖，在讓CSP去執行，因為Assignment 2 我們是擺好數字，然後去猜測是否為地雷，所以可以是complete的，而如果今天要應用在這次的遊戲上，應該在一開始使用Logic先去探索，然後探索到一定的程度的時候，在去使用CSP去給值，或是遇到Stuck的時候，這時候沒有辦法在使用Logic，所以我們再去使用CSP猜也可以，因為其實Logic跑的速度會比CSP還要快，但老師上課有說這個問題更適合CSP，因為應該是可以去找到結果的。

實驗中的遇到的問題：

1. 在做Safe Cell數量的時候，一開始實驗忘記地圖的變因要固定，所以不小心做出結果發現，沒有顯著的改變，後來發現原因是因為每次實驗地圖都會變，所以才會導致如此結果，後來固定好地圖的變因之後，因為原先設定的Safe Cell是每次都去取，所以Safe Cell = 5並不會等於Safe Cell = 4 再去取一顆，所以這裡的變因又會導致實驗的誤差太大，最終控制地圖變因、Safe Cell變因最後得到結果。過程失敗的圖：

Test 失敗第一次：(地圖會變、Safe Cell會變) 無顯著的變化，實驗失敗。

Test 失敗第二次：(地圖不變、Safe Cell會變) 經由圖，無法看出明顯的變化，實驗失敗。

1. Stuck的解決方法，在第四頁的圖四中，這邊的Stuck較容易解決，只要透過判斷是否mine都已經被給值了，那麼這邊只要去偵查-1的八個direction，去給值就可以達到，而我們常在Hard中遇到的Stuck會比這個更困難，這裡是因為Safe Cell給的值太過於密集於一區，而導致此情況，例如：

在這方面的解決，我目前覺得只有兩種方式，就是上面Discussion 3提及去猜測，或是使用CSP去猜測，但是在這方面我有看到網路上有些論文說minesweeper是dead-end-tree，因為在這種玩法下，必須要去猜測而非會有準確的答案(像是Assignment#2)。

Future investigation：原本還想探討控制**地雷數目=round(sqrt(R\*C))地雷數的增加會導致stuck的情況發生更為頻繁嗎？**我認為答案應該是Yes，但是因為寫的頁數不夠了，所以這邊放在Future investigation去探討。

Reference：

Stuck Game：  
<https://puzzling.stackexchange.com/questions/364/how-do-i-approach-a-stuck-game-of-minesweeper>

程式碼(C++) 使用Dev C++

1. #include<iostream>
2. #include<string.h>
3. #include<stdio.h>
4. #include<stdlib.h>
5. #include<time.h>
6. #include<math.h>
7. #include<iomanip>
8. #include<vector>
9. #include <windows.h>
11. using namespace std;
13. */\**
14. *author : 蔡怡君 0613413*
15. *content : Minesweeper solution*
16. *using propositional logic*
17. *\*/*
19. // mine Map Small / Medium / Big
20. int MAP[16][30]={0}, R, C, num\_mines ;
21. int GameMAP[16][30];
23. struct point
24. {
25. int x,y;
26. int hint;
27. bool mine;
28. }dir[8];
30. vector<point>hint; // 裏頭有炸彈的位置
31. //vector<point>safe;
33. struct cell
34. {
35. int x,y;
36. int hint;
37. int sign; // 看是-A 還是 A
38. bool mine;
39. };
41. bool operator == (const cell &p1,const cell &p2)
42. {
43. if(p1.x == p2.x && p1.y == p2.y)
44. return true;
45. else
46. return false;
47. }
49. class clause
50. {
51. public:
52. vector<cell>element;
53. int getn();
54. clause operator+(const clause&);
55. bool operator==(const clause&);
56. bool entail(const clause&);
57. int cpm(const clause&);
58. void print();
59. };
61. int clause::getn()
62. {
63. return element.size();
64. }
66. bool clause::operator==(const clause &B)
67. {
68. if(element.size() != B.element.size())return false;
69. int index = 0,i;
70. bool flag = false;
71. while( index < element.size() ){
72. for(i = 0 ; i < B.element.size() ; i ++){
73. if(element[index].x == B.element[i].x && element[index].y == B.element[i].y)
74. if(element[index].sign == B.element[i].sign)
75. break;
76. }
77. if(i == B.element.size()) // 代表跑完了都還沒找到
78. return false;
79. index ++;
80. }
81. return true;
82. */\*不檢查符號是否相符 因為外面抵銷也有用到這個*
83. *if(element[index].sign == B.element[i].sign)*
84. *\*/*
85. }
87. clause clause::operator+(const clause &B)
88. {
89. clause tmp;
90. // 把 this 的cell 都給 tmp
91. int j,fi,fj;;
92. bool flag;
93. for(int i = 0 ; i < (\*this).getn() ; i++){
94. flag = false;
95. for(j = 0 ; j < B.element.size() ; j ++){
96. if( (\*this).element[i] == B.element[j] )
97. if( (\*this).element[i].sign == -B.element[j].sign ){
98. fi = i; fj = j;
99. flag = true;
100. break;
101. }
102. }
103. if(flag)
104. break;
105. }
107. for(int i = 0 ; i < (\*this).getn(); i ++){
108. if(i == fi) continue;
109. tmp.element.push\_back( (\*this).element[i] );
110. }
112. for(int i = 0; i < B.element.size(); i++){
113. if( i == fj) continue;
114. tmp.element.push\_back(B.element[i]);
115. }
117. // 如果句子產生 +A,+A - >> 刪除其中一個
118. if(tmp.getn() == 2 && tmp.element[0] == tmp.element[1])
119. tmp.element.erase(tmp.element.begin()+1);
121. return tmp;
122. }
124. // 查看 complementary 幾個句子
125. int clause::cpm(const clause &B)
126. {
127. int count = 0;
128. int index = 0,i;
129. while( index < element.size() ){
130. for(i = 0 ; i < B.element.size() ; i ++){
131. if(element[index].x == B.element[i].x && element[index].y == B.element[i].y)
132. if(element[index].sign == -B.element[i].sign)
133. count++;
134. }
135. index ++;
136. }
137. return count;
138. }
140. // 看this是否entail B 比較嚴謹？
141. bool clause::entail(const clause &B)
142. {
143. if(element.size()==0) return false;
144. if(element.size() < B.element.size()){
145. int index = 0,i;
146. while( index < element.size() ){
147. for(i = 0 ; i < B.element.size() ; i ++){
148. if(element[index].x == B.element[i].x && element[index].y == B.element[i].y)
149. if(element[index].sign == B.element[i].sign)
150. break;
151. }
152. if(i == B.element.size()) // 代表跑完了都還沒找到
153. return false;
154. index ++;
155. }
156. // 如果 this本來長度就比較小 且in B裡頭都有找到的話 代表 this entail B
157. return true;
158. }
159. return false;
160. }
162. void clause::print()
163. {
164. for(int i = 0 ; i < element.size(); i ++){
165. if(element[i].sign == -1)
166. cout<<"-";
167. else if(element[i].sign == 1)
168. cout<<"+";
169. cout<<"("<<element[i].x<<","<<element[i].y<<") ";
170. }
171. //cout<<endl;
172. }

175. vector<clause> KB;
176. vector<clause> KB0;
178. void initial()
179. {
180. dir[0]={-1,-1};
181. dir[1]={-1,0};
182. dir[2]={-1,1};
183. dir[3]={0,-1};
184. dir[4]={0,1};
185. dir[5]={1,-1};
186. dir[6]={1,0};
187. dir[7]={1,1};
189. // game
190. for(int i = 0 ; i < R ; i ++)
191. for(int j = 0 ; j < C ; j ++)
192. GameMAP[i][j] = -1;
193. KB.clear();
194. KB0.clear();
195. }
197. bool in(vector<clause> K,clause c)
198. {
199. for(int i = 0 ; i < K.size() ; i ++){
200. if(K[i] == c)
201. return true;
202. }
203. return false;
204. }
206. bool in(vector<clause> K,int index,clause c)
207. {
208. // 看它的前面有沒有相同的
209. for(int i = 0 ; i < index ; i ++){
210. if(K[i] == c)
211. return true;
212. }
213. return false;
214. }
216. bool isVaild(int i,int j)
217. {
218. return ( (i >= 0 && i < R ) && (j >= 0 && j < C) );
219. }
221. // ----------- control mode start ------------
222. // - create map
223. void mode(int mode)
224. {
225. int mine = 0;
226. // control
227. srand( time(NULL) );
228. if(mode == 1){
229. R = 9; C = 9;
230. num\_mines = 10;
231. while( mine < num\_mines){
232. int x = rand()%R;
233. int y = rand()%C;
234. if(MAP[x][y] == 0){
235. MAP[x][y] = -1; // mean mines
236. mine++;
237. point h = {x,y,MAP[x][y],true};
238. hint.push\_back(h);
239. }
240. }
241. }
242. else if(mode == 2){
243. R = 16; C = 16;
244. num\_mines = 25;
245. while( mine < num\_mines){
246. int x = rand()%R;
247. int y = rand()%C;
248. if(MAP[x][y] == 0){
249. MAP[x][y] = -1; // mean mines
250. mine++;
251. point h = {x,y,MAP[x][y],true};
252. hint.push\_back(h);
253. }
254. }
255. }
256. else{
257. R = 16; C =30;
258. num\_mines = 99;
259. while( mine < num\_mines){
260. int x = rand()%R;
261. int y = rand()%C;
262. if(MAP[x][y] == 0){
263. MAP[x][y] = -1; // mean mines
264. mine++;
265. point h = {x,y,MAP[x][y],true};
266. hint.push\_back(h);
267. }
268. }
269. }
271. // game
272. for(int i = 0 ; i < R ; i ++)
273. for(int j = 0 ; j < C ; j ++)
274. GameMAP[i][j] = -1;
275. }
277. void fillin(vector<point> hhint)
278. {
279. // 去填 Hint 周圍的值
280. for(int i = 0 ; i < hhint.size() ; i ++){
281. // check hint 的八維
282. for(int j = 0 ; j < 8 ; j ++){
283. int px = hhint[i].x + dir[j].x;
284. int py = hhint[i].y + dir[j].y;
285. if(isVaild(px , py) && MAP[px][py] != -1){
286. MAP[px][py] += 1;
287. }
288. }
289. }
290. }
292. //---- using for test
293. vector<clause>first;
294. void put\_safe\_cell(int safe)
295. {
296. // \*\*\* 去 input initial safe cells 可透過去變這裡去分析
297. int i = 0;
298. while(i < safe){
299. int x = rand()%R , y = rand()%C;
300. // 不是炸彈
301. if(MAP[x][y] != -1){
302. cell s ={x,y,MAP[x][y],-1,false};
303. clause tmp;
304. tmp.element.push\_back(s);
305. if(!in(first,tmp)){
306. first.push\_back(tmp);
307. //KB.push\_back(tmp);
308. i++;
309. //cout<<"x:"<<x<<" y:"<<y<<endl;
310. }
311. }
312. }
314. }
316. void test\_safe(int add)
317. {
318. int i = 0;
319. while( i < add){
320. int x = rand()%R , y = rand()%C;
321. // 不是炸彈
322. if(MAP[x][y] != -1){
323. cell s ={x,y,MAP[x][y],-1,false};
324. clause tmp;
325. tmp.element.push\_back(s);
326. if(!in(first,tmp)){
327. first.push\_back(tmp);
328. i++;
329. }
330. }
331. }
332. }
334. // ----------- control mode end ------------
336. // ----------- game mode start ------------
338. bool check\_dub\_sub(clause now)
339. {
340. // 第一次判斷 for case 3
341. bool first = true;
342. for(int j = 0 ; j < KB.size() ; j++){
343. //case 1 檢查 KB裡面會不會有same的？ same就跳過
344. if(KB[j] == now){
345. return true;
346. }
347. //case 2檢查 subsumption
348. if(KB[j].entail(now)){
349. return true;
350. }
351. //case 3去看說 B是否entail this 如果是True 原本的要刪除 新增這個進去
352. if(now.entail(KB[j])){
353. if(first){ //第 一次遇到 assign直接取代原本的
354. KB[j] = now;
355. first = false;
356. }
357. else{
358. KB.erase(KB.begin()+j);
359. j--;
360. }
361. }
362. }
363. if(!first)
364. return true;
365. return false;
366. }
368. // 印出所有的KB0
369. void single\_all\_in\_clause(vector<clause> A)
370. {
371. for(int i = 0 ; i < A.size() ; i ++){
372. if(A[i].getn() == 1)
373. cout<<A[i].element[0].sign<<"("<<A[i].element[0].x<<","<<A[i].element[0].y<<")";
374. }
375. cout<<endl;
376. }
378. // 印出所有的KB clause
379. void all\_in\_clause(vector<clause> A)
380. {
381. for(int i = 0 ; i < A.size() ; i ++){
382. for(int j = 0 ; j < A[i].getn(); j++)
383. cout<<A[i].element[j].sign<<"("<<A[i].element[j].x<<","<<A[i].element[j].y<<")";
384. cout<<endl;
385. }
386. }
388. void pretty\_print(int m, vector<cell> local, int combo, int sign)
389. {
390. clause now;
391. for (int i = 0; i < m; ++i) {
392. if ((combo >> i) & 1){
393. if(sign == 1) { local[i].sign = 1; } //cout<<"+";
394. else if(sign == -1) { local[i].sign = -1; } //cout<<"-";
395. // 不用檢查 KB0 因為 KB0 = true的話早就被給值了
396. now.element.push\_back(local[i]);
397. //cout<<"("<< local[i].x <<","<<local[i].y<<")"<< ' ';
398. }
399. if( i == m-1){
400. if(!check\_dub\_sub(now))
401. KB.push\_back(now);
402. }
403. }
404. //cout << endl;
405. }
407. void combination(int m, int n, vector<cell>local, int sign)
408. {
409. int combo = (1 << n) - 1; // k bit sets
410. while (combo < 1<<m) {
411. pretty\_print(m,local,combo,sign);
413. int x = combo & -combo;
414. int y = combo + x;
415. int z = (combo & ~y);
416. combo = z / x;
417. combo >>= 1;
418. combo |= y;
419. }
420. }
422. int go\_single=0,go\_nonsingle=0;
423. void game\_mode()
424. {
425. int total\_mines = 0;
426. int stucktime = 0;
427. //num\_mines
428. while(total\_mines <= num\_mines){
429. // if there's a single-lateral clause
430. bool single = false;
431. for(int i = 0 ; i < KB.size() ; i ++){
432. if(KB[i].getn() == 1){
433. single = true; go\_single++;
434. clause current = KB[i];
435. int cx = current.element[0].x;
436. int cy = current.element[0].y;
437. if(current.element[0].sign == -1){
438. current.element[0].mine = false; // 沒有地雷
439. current.element[0].hint = MAP[cx][cy];
440. GameMAP[cx][cy] = MAP[cx][cy]; // 放hint值
441. }
442. else if (current.element[0].sign == 1){
443. current.element[0].mine = true; // 有地雷
444. current.element[0].hint = MAP[cx][cy];
445. GameMAP[cx][cy] = 666; //means 有地雷
446. total\_mines++;
447. }
448. KB.erase(KB.begin()+i);
449. //i--;
450. KB0.push\_back(current);
451. // 一對多 matching 新clause判斷完的 & 其他的KB句子 對消
452. for(int a = 0 ; a < KB.size(); a ++){
453. // 如果現在這個current entail KB裡的 或是 = 就刪除
454. if( current.entail(KB[a]) || current == KB[a] ){
455. // 刪除這個 KB[a]
456. KB.erase(KB.begin()+a);
457. a--;
458. continue;
459. }
460. for(int b = 0 ; b < KB[a].getn() ; b ++){
461. if(KB[a].element[b] == current.element[0] && KB[a].getn()!= 1)
462. // 抵銷
463. if(KB[a].element[b].sign == - current.element[0].sign){
464. KB[a].element.erase(KB[a].element.begin()+b);
465. // 去檢查新的句子有沒有在 KB0有的話 刪除
466. if( in(KB0,KB[a]) ){
467. KB.erase(KB.begin()+a);
468. a--;
469. }
470. // 去檢查新的句子有沒有在KB(index = a之前)裡有重複的！(重要)
471. else if(in(KB,a,KB[a])){
472. KB.erase(KB.begin()+a);
473. a--;
474. }
475. break;
476. }
477. }
479. }
480. // if the cell is safe
481. if(!current.element[0].mine){
482. // query game control
483. cell now = current.element[0];
484. vector<cell> todo;
485. int local\_mine = now.hint , unmarked = 0;
486. // 跑鄰居 是否有炸彈 、 marked
487. for(int a = 0 ; a < 8 ; a ++){
488. int tmpx = now.x + dir[a].x ;
489. int tmpy = now.y + dir[a].y;
490. if(isVaild(tmpx, tmpy)){
491. if(GameMAP[tmpx][tmpy] == 666){ // means有地雷
492. local\_mine--;
493. }
494. if(GameMAP[tmpx][tmpy] == -1){ // unmarked
495. cell tmp = {tmpx,tmpy};
496. unmarked ++;
497. todo.push\_back(tmp);
498. }
499. }
500. }
501. if(local\_mine == 0 && unmarked == 0) continue;
502. if(local\_mine == unmarked && unmarked !=0){ // n = m
503. //cout<<"positive:";
504. while(!todo.empty()){
505. clause positive;
506. cell po = todo.back();
507. todo.pop\_back();
508. po.sign = 1;
509. positive.element.push\_back(po);
510. if(!in(KB,positive) && !in(KB0,positive)){
511. //cout<<"O";
512. KB.push\_back(positive);
513. }
514. //else cout<<"X";
515. //cout<<"+"<<"("<<po.x<<","<<po.y<<") ";
516. }
517. //cout<<endl;
518. }
519. if(local\_mine == 0){ // n = 0
520. //cout<<"negative:"<<endl;
521. while(!todo.empty()){
522. clause negative;
523. cell ne = todo.back();
524. todo.pop\_back();
525. ne.sign = -1;
526. negative.element.push\_back(ne);
527. if(!in(KB,negative) && !in(KB0,negative)){
528. //cout<<"O";
529. KB.push\_back(negative);
530. }
531. //else cout<<"X";
532. //cout<<"-"<<"("<<ne.x<<","<<ne.y<<") ";
533. }
534. //cout<<endl;
535. }
536. if(unmarked > local\_mine && local\_mine > 0){ // m > n > 0
537. combination(unmarked,unmarked-local\_mine+1,todo,1);
538. combination(unmarked,local\_mine+1,todo,-1);
539. //todo.clear();
540. }
541. }
542. break;
543. }
544. }
545. // global constraints
546. //if(go\_single > R\*C - round(sqrt(R\*C))){
547. //combination();
548. //}
549. bool make\_single = false;
550. if(!single && stucktime == 10)
551. break;
552. // 沒有single-lateral clause
553. if(!single){
554. go\_nonsingle ++;
555. int Before = KB.size();
556. // pairwise "matching" of all clause 多不多
557. if(KB.size() == 0) break;
558. for(int i = 0 ; i < KB.size()-1 ; i ++){
559. for(int j = i + 1 ; j < KB.size() ; j ++){
560. // cpm 看complementary 句子有幾個
561. if( KB[i].cpm(KB[j]) == 1 && KB[i].getn() <= 2 && KB[j].getn() <= 2){
562. //cout<<"i:"<<i<<" j:"<<j<<" size:"<<KB.size()<<endl;
563. clause tmp = KB[i] + KB[j];
565. if(tmp.element.size() == 1)
566. make\_single = true;
568. // no duplication & subsumption
569. if(tmp.element.size()!=0 && !check\_dub\_sub(tmp))
570. {
571. KB.push\_back(tmp);
572. }
573. }
574. }
575. }
576. int After = KB.size();
577. if(Before == After && make\_single == false)
578. stucktime ++;
579. //cout<<"-------------------Bye---------------------";
580. }
581. }
582. }

585. // ----------- game mode end ------------
586. void setColor(int color);
587. void print\_map()
588. {
589. for(int i = 0 ; i < R ;i ++){
590. for(int j = 0 ; j < C ; j ++){
591. if(MAP[i][j] == -1){
592. setColor(12);
593. cout<<setw(2)<<"\*"<<" ";
594. }
595. else if(MAP[i][j] != 0){
596. setColor(7); //預設
597. cout<<setw(2)<<MAP[i][j]<<" ";
598. }
599. else
600. cout<<" ";
601. }
602. cout<<endl;
603. }
604. }
606. void print\_GameMAP()
607. {
608. for(int i = 0 ; i < R ;i ++){
609. for(int j = 0 ; j < C ; j ++){
610. if(GameMAP[i][j] == 666){
611. setColor(12);
612. cout<<setw(2)<<"\*"<<" ";
613. }
614. else if(GameMAP[i][j] == -1){
615. setColor(240);
616. cout<<GameMAP[i][j]<<" ";
617. setColor(7);
618. }
619. else if(GameMAP[i][j] != 0){
620. setColor(7);
621. cout<<setw(2)<<GameMAP[i][j]<<" ";
622. }
623. else
624. cout<<" ";
625. }
626. cout<<endl;
627. }
628. }

631. int main()
632. {
633. int level;
634. time\_t start,end;
635. double t;
636. cout << "LeveL :1.Easy 2.Medium 3.Hard" << endl;
637. cout << "plz enter level:";
638. cin >> level;
640. // control mode put value to 8 direcction
641. initial();
642. mode(level); // set mode
643. fillin(hint); // fill in the neibor of hint
645. // 最一開始
646. put\_safe\_cell(round(sqrt(R\*C))-4);
647. for(int a = 0; a < 15 ; a ++){
648. test\_safe(1);
649. // 歸零
650. initial();
651. for(int j = 0 ; j < first.size() ; j++)
652. KB.push\_back(first[j]);
653. //cout<<KB.size()<<endl;
654. // game mode
655. start = clock();
656. game\_mode();
657. end = clock();
658. t = ((double)(end-start))/CLOCKS\_PER\_SEC;
659. printf("%f**\t**",t);
660. cout<<endl;
661. //printf("Time : %fs\n",t);
662. }
664. cout<<"-------------- control mode ---------------**\n**";
665. print\_map(); // print it out
666. cout<<endl;
667. cout<<"-------------- Game mode ---------------**\n**";
668. print\_GameMAP();
669. cout<<endl;
670. setColor(242);
671. cout<<"single判斷:"<<go\_single<<" nonSingle判斷:"<<go\_nonsingle<<endl;
672. return 0;
673. }
675. // 控制炸彈顏色
676. void setColor(int color){
677. HANDLE hConsole;
678. hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
679. SetConsoleTextAttribute(hConsole,color);
680. }