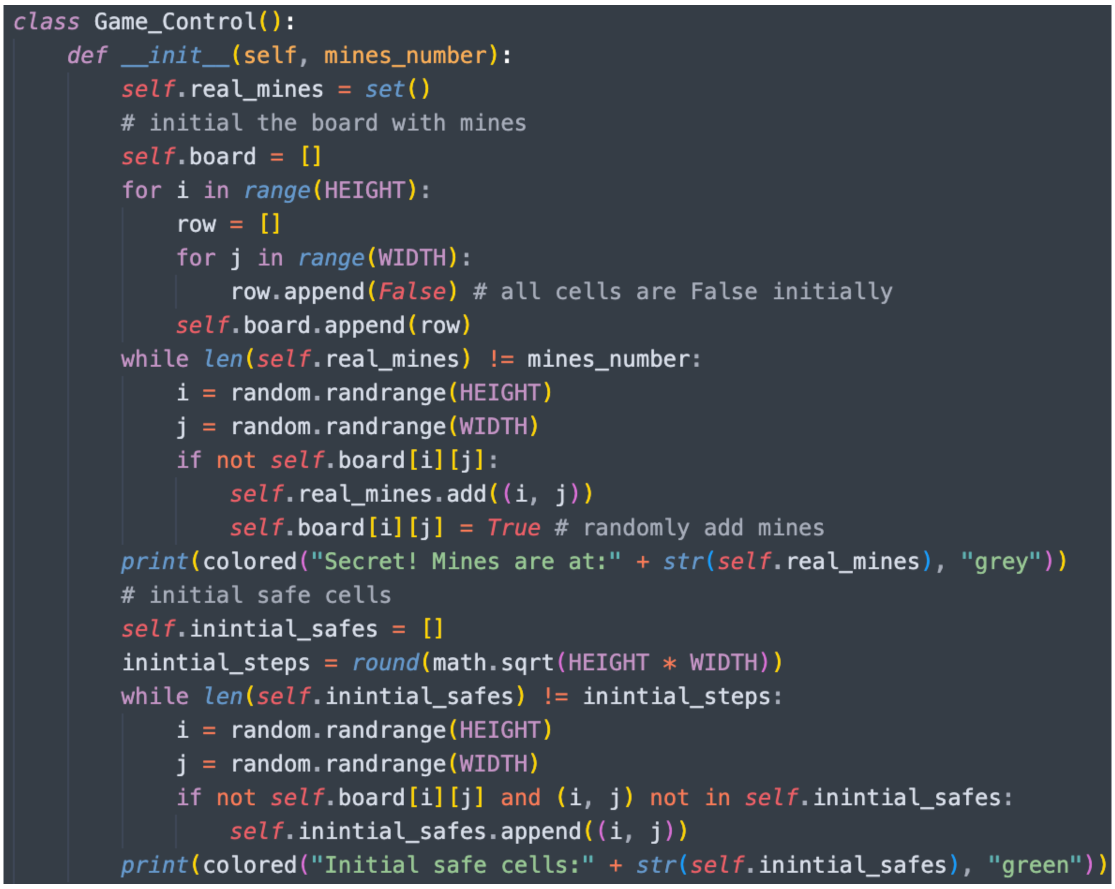
AI Capstone NYCU Spr2023 Assignment #3

陳存佩 109550171

此 Report 將依 Spec 大標跟程式截圖與註解說明，並附上實驗結果與心得

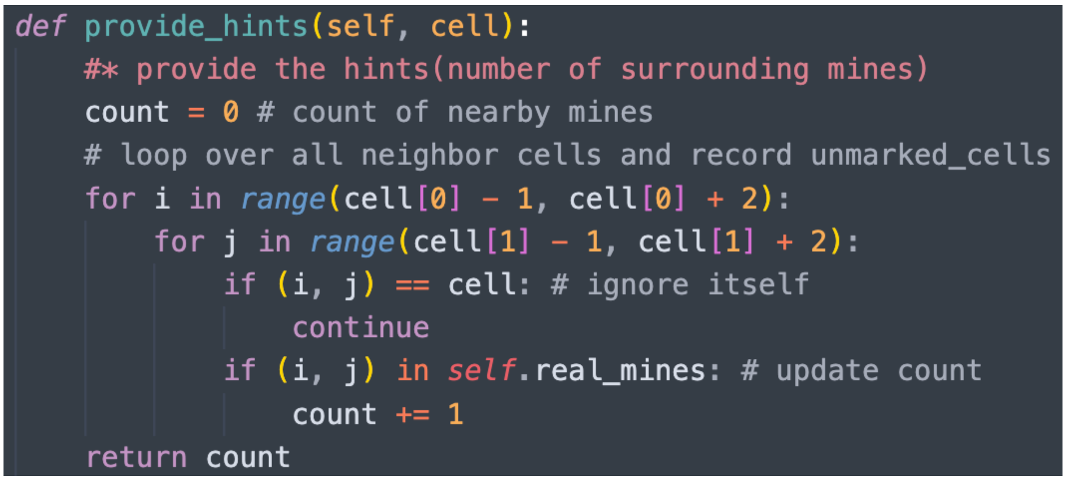
Tasks of the game control module:

Game control 的部分使用 class Game_Control()來控制，在 init 的時候初始化板子跟地雷，還有提供 initial safe cells



```python
class Game_Control():
    def __init__(self, mines_number):
        self.real_mines = set()
        # initial the board with mines
        self.board = []
        for i in range(HEIGHT):
            row = []
            for j in range(WIDTH):
                row.append(False) # all cells are False initially
            self.board.append(row)
        while len(self.real_mines) != mines_number:
            i = random.randrange(HEIGHT)
            j = random.randrange(WIDTH)
            if not self.board[i][j]:
                self.real_mines.add((i, j))
                self.board[i][j] = True # randomly add mines
        print(colored("Secret! Mines are at:" + str(self.real_mines), "grey"))
        # initial safe cells
        self.inintial_safes = []
        inintial_steps = round(math.sqrt(HEIGHT * WIDTH))
        while len(self.inintial_safes) != inintial_steps:
            i = random.randrange(HEIGHT)
            j = random.randrange(WIDTH)
            if not self.board[i][j] and (i, j) not in self.inintial_safes:
                self.inintial_safes.append((i, j))
        print(colored("Initial safe cells:" + str(self.inintial_safes), "green"))
```

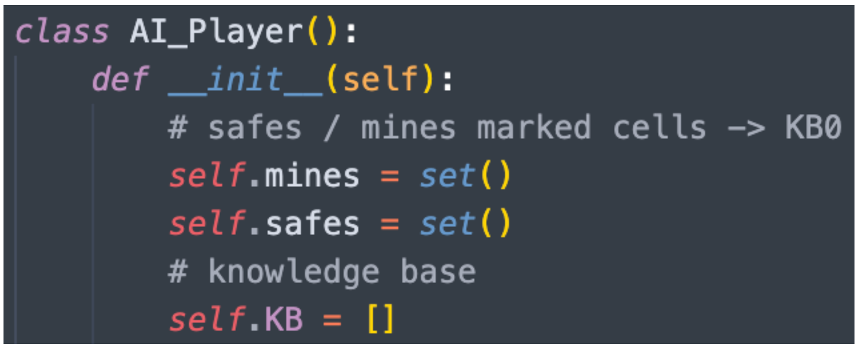
每次當 player 新找到一個 safe cell，就觸發 provide_hints 給提示



```python
def provide_hints(self, cell):
    #* provide the hints(number of surrounding mines)
    count = 0 # count of nearby mines
    # loop over all neighbor cells and record unmarked_cells
    for i in range(cell[0] - 1, cell[0] + 2):
        for j in range(cell[1] - 1, cell[1] + 2):
            if (i, j) == cell: # ignore itself
                continue
            if (i, j) in self.real_mines: # update count
                count += 1
    return count
```

Tasks of the player module:

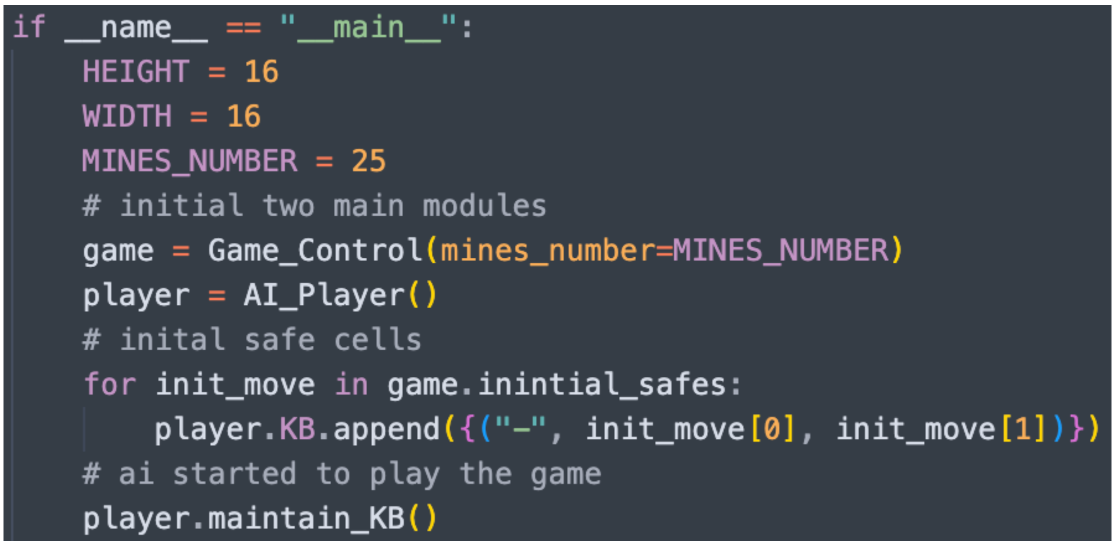
Play 的部分是 class AI_Player 處理，這也是此作業最複雜的部分，最主要是透過 maintain_KB()裡面的 while 迴圈來處理（詳細流程將在 Game flow 描述），在 init 的時候初始化 mines 跟 safes 兩個 set 來記錄被玩家標記的 cell，也就是 spec 裡面的 KB0



```python
class AI_Player():
    def __init__(self):
        # safes / mines marked cells -> KB0
        self.mines = set()
        self.safes = set()
        # knowledge base
        self.KB = []
```

Game flow:

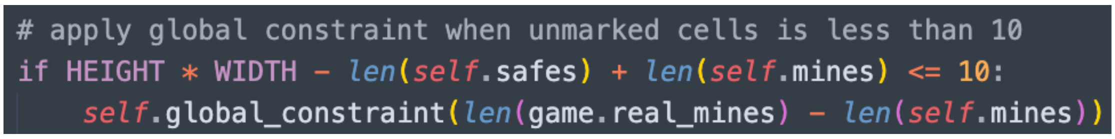
遊戲初始化：宣告兩個主要的 class，並將 initial safe cells 加入 KB



```python
if __name__ == "__main__":
    HEIGHT = 16
    WIDTH = 16
    MINES_NUMBER = 25
    # initial two main modules
    game = Game_Control(mines_number=MINES_NUMBER)
    player = AI_Player()
    # inital safe cells
    for init_move in game.inintial_safes:
        player.KB.append({("-", init_move[0], init_move[1])})
    # ai started to play the game
    player.maintain_KB()
```

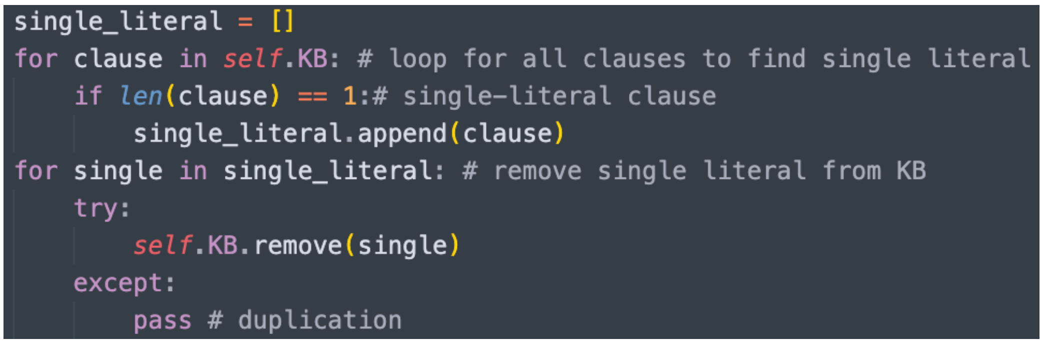
開始遊戲：進入 maintain_KB 函式，每次迭代會進行下列動作：

1. 決定是否新增 global constraint clauses



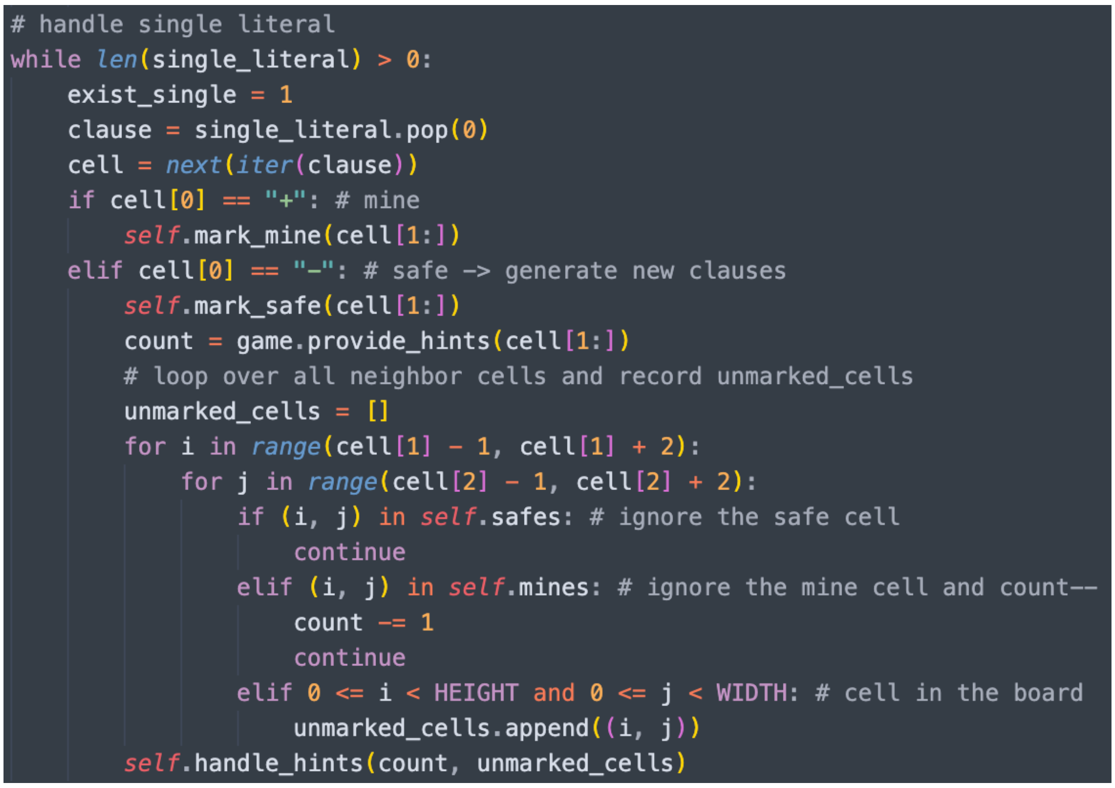
```python
# apply global constraint when unmarked cells is less than 10
if HEIGHT * WIDTH - len(self.safes) + len(self.mines) <= 10:
    self.global_constraint(len(game.real_mines) - len(self.mines))
```

2. 找出所有 single literal clause，將其從 KB 移除

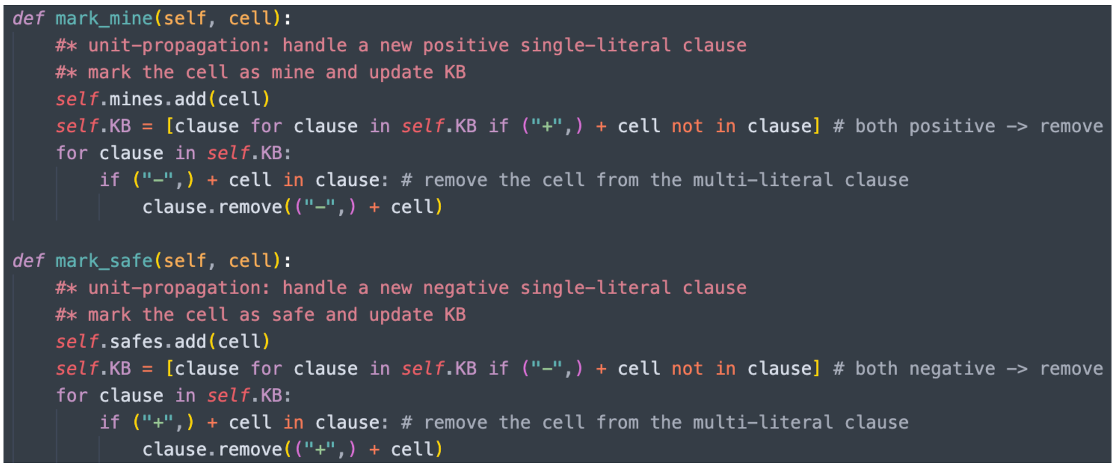


```python
single_literal = []
for clause in self.KB: # loop for all clauses to find single literal
    if len(clause) == 1:# single-literal clause
        single_literal.append(clause)
for single in single_literal: # remove single literal from KB
    try:
        self.KB.remove(single)
    except:
        pass # duplication
```

3. 將 single literal 標記為地雷或安全並加入 KB0，並進行 unit propagation，接著 safe cell 進入 handle_hints()，此函式將產生新的 clause 且檢查後插入

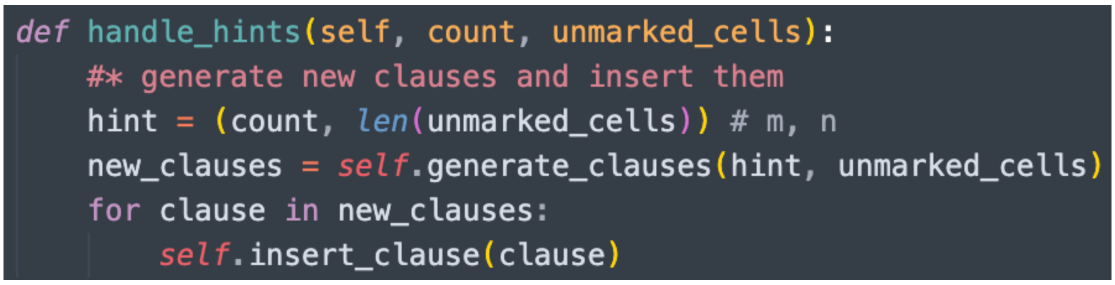


```python
# handle single literal
while len(single_literal) > 0:
    exist_single = 1
    clause = single_literal.pop(0)
    cell = next(iter(clause))
    if cell[0] == "+": # mine
        self.mark_mine(cell[1:])
    elif cell[0] == "-": # safe -> generate new clauses
        self.mark_safe(cell[1:])
        count = game.provide_hints(cell[1:])
        # loop over all neighbor cells and record unmarked_cells
        unmarked_cells = []
        for i in range(cell[1] - 1, cell[1] + 2):
            for j in range(cell[2] - 1, cell[2] + 2):
                if (i, j) in self.safes: # ignore the safe cell
                    continue
                elif (i, j) in self.mines: # ignore the mine cell and count--
                    count -= 1
                    continue
                elif 0 <= i < HEIGHT and 0 <= j < WIDTH: # cell in the board
                    unmarked_cells.append((i, j))
        self.handle_hints(count, unmarked_cells)
```



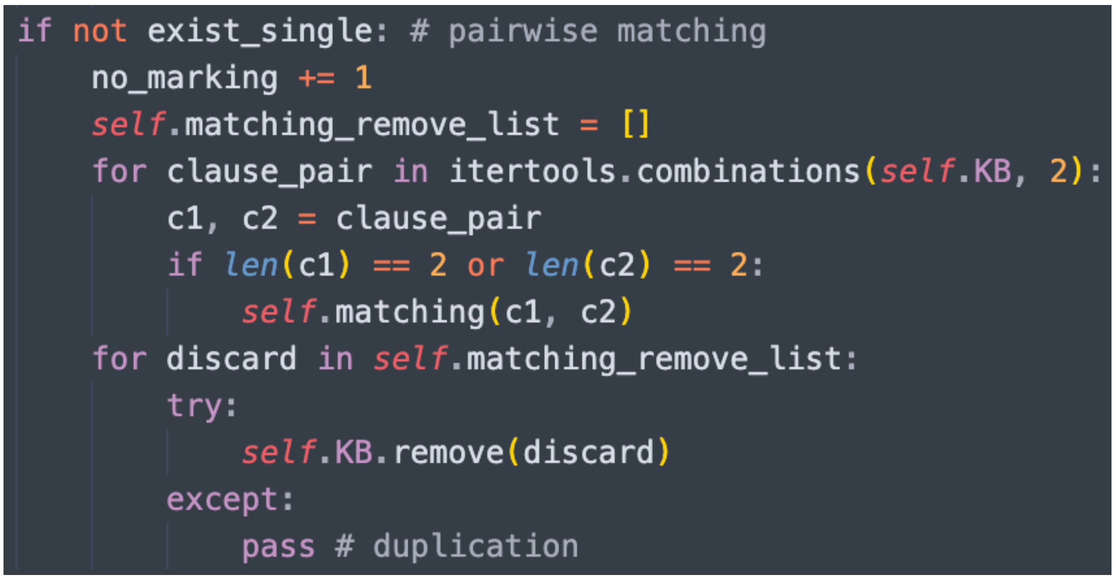
```python
def mark_mine(self, cell):
    #* unit-propagation: handle a new positive single-literal clause
    #* mark the cell as mine and update KB
    self.mines.add(cell)
    self.KB = [clause for clause in self.KB if ("+",) + cell not in clause] # both positive -> remove
    for clause in self.KB:
        if ("-",) + cell in clause: # remove the cell from the multi-literal clause
            clause.remove(("-",) + cell)

def mark_safe(self, cell):
    #* unit-propagation: handle a new negative single-literal clause
    #* mark the cell as safe and update KB
    self.safes.add(cell)
    self.KB = [clause for clause in self.KB if ("-",) + cell not in clause] # both negative -> remove
    for clause in self.KB:
        if ("+",) + cell in clause: # remove the cell from the multi-literal clause
            clause.remove(("+",) + cell)
```



```python
def handle_hints(self, count, unmarked_cells):
    #* generate new clauses and insert them
    hint = (count, len(unmarked_cells)) # m, n
    new_clauses = self.generate_clauses(hint, unmarked_cells)
    for clause in new_clauses:
        self.insert_clause(clause)
```

4. 如果沒有任何新的 single literal，所有 KB 內的 clause 兩兩配對進行 matching（只有長度為 2 才做 matching）

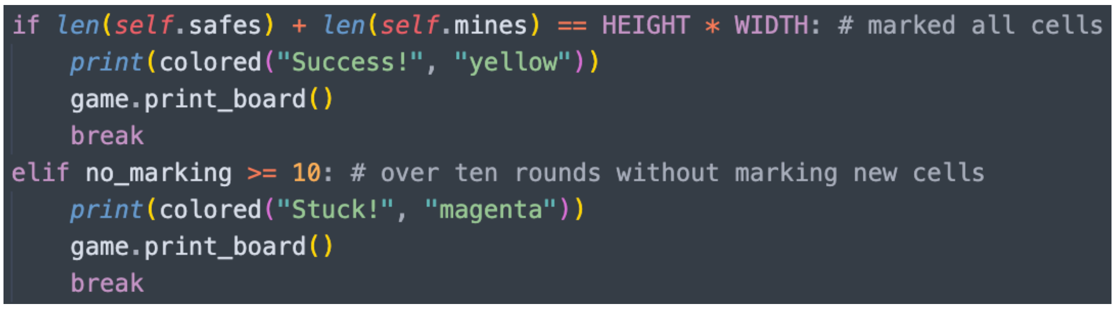


```python
if not exist_single: # pairwise matching
    no_marking += 1
    self.matching_remove_list = []
    for clause_pair in itertools.combinations(self.KB, 2):
        c1, c2 = clause_pair
        if len(c1) == 2 or len(c2) == 2:
            self.matching(c1, c2)
    for discard in self.matching_remove_list:
        try:
            self.KB.remove(discard)
        except:
            pass # duplication
```

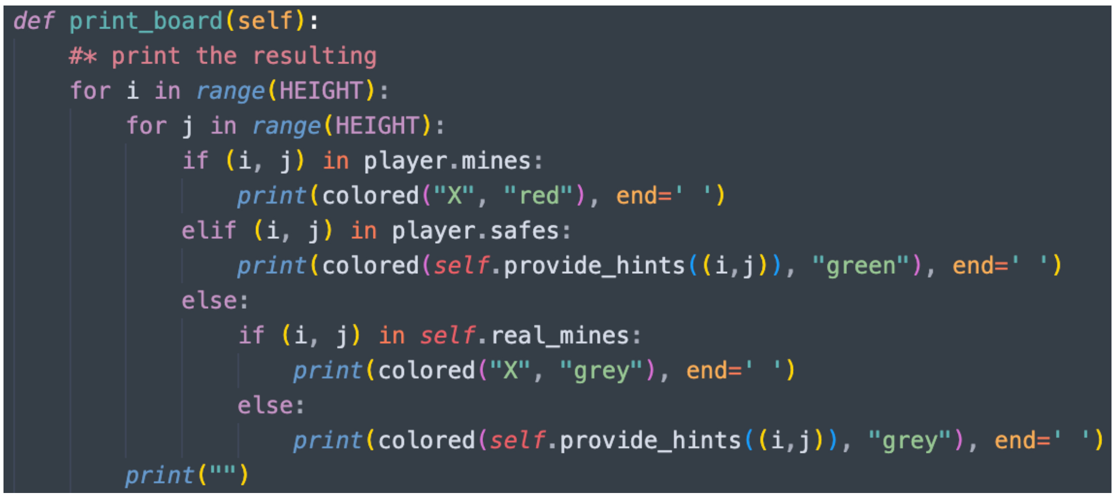
5. 檢查程式是否達終止條件，並印出最終結果

Game termination:

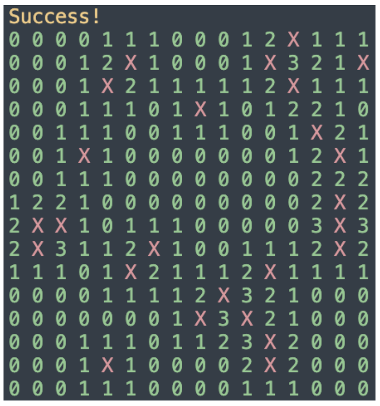
成功的條件是所有 cell 都被標記完成，卡住的的條件是超過十輪都沒有新的標記產生，當達到終止條件，就結束 maintain_KB 的迴圈，Game_Control 印出最終的板子狀態，這邊透過 termcolor 的 colored 函式讓輸出有顏色，方便辨識



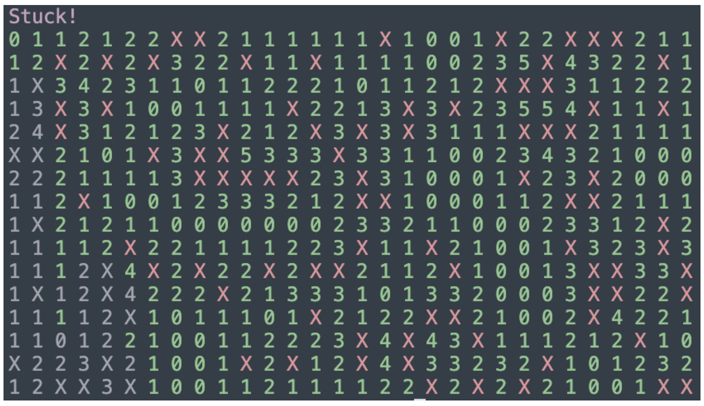
```python
if len(self.safes) + len(self.mines) == HEIGHT * WIDTH: # marked all cells
    print(colored("Success!", "yellow"))
    game.print_board()
    break
elif no_marking >= 10: # over ten rounds without marking new cells
    print(colored("Stuck!", "magenta"))
    game.print_board()
    break
```



```python
def print_board(self):
    #* print the resulting
    for i in range(HEIGHT):
        for j in range(HEIGHT):
            if (i, j) in player.mines:
                print(colored("X", "red"), end=' ')
            elif (i, j) in player.safes:
                print(colored(self.provide_hints((i,j)), "green"), end=' ')
            else:
                if (i, j) in self.real_mines:
                    print(colored("X", "grey"), end=' ')
                else:
                    print(colored(self.provide_hints((i,j)), "grey"), end=' ')
        print("")
```



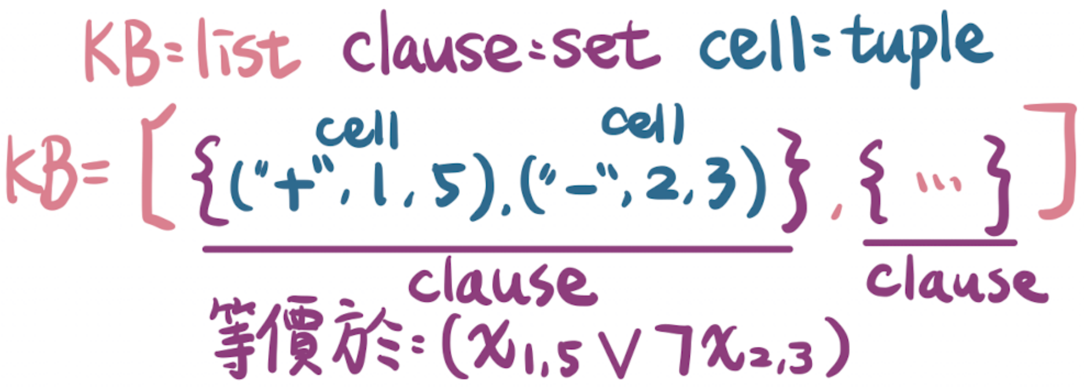
```
Success!
0 0 0 0 1 1 1 0 0 0 1 2 X 1 1 1
0 0 0 1 2 X 1 0 0 0 1 X 3 2 1 X
0 0 0 1 X 2 1 1 1 1 1 2 X 1 1 1
0 0 0 1 1 1 0 1 X 1 0 1 2 2 1 0
0 0 1 1 1 0 0 1 1 1 0 0 1 X 2 1
0 0 1 X 1 0 0 0 0 0 0 0 1 2 X 1
0 0 1 1 1 0 0 0 0 0 0 0 0 2 2 2
1 2 2 1 0 0 0 0 0 0 0 0 0 2 X 2
2 X X 1 0 1 1 1 0 0 0 0 0 3 X 3
2 X 3 1 1 2 X 1 0 0 1 1 1 2 X 2
1 1 1 0 1 X 2 1 1 1 2 X 1 1 1 1
0 0 0 0 1 1 1 1 2 X 3 2 1 0 0 0
0 0 0 0 0 0 0 1 X 3 X 2 1 0 0 0
0 0 0 1 1 1 0 1 1 2 3 X 2 0 0 0
0 0 0 1 X 1 0 0 0 0 2 X 2 0 0 0
0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0
```



```
Stuck!
0 1 1 2 1 2 2 X X 2 1 1 1 1 1 1 X 1 0 0 1 X 2 2 X X X 2 1 1
1 2 X 2 X 2 X 3 2 2 X 1 1 X 1 1 1 1 0 0 2 3 5 X 4 3 2 2 X 1
1 X 3 4 2 3 1 1 0 1 1 2 2 2 1 0 1 1 2 1 2 X X X 3 1 1 2 2 2
1 3 X 3 X 1 0 0 1 1 1 1 X 2 2 1 3 X 3 X 2 3 5 5 4 X 1 1 X 1
2 4 X 3 1 2 1 2 3 X 2 1 2 X 3 X 3 X 3 1 1 1 X X X 2 1 1 1 1
X X 2 1 0 1 X 3 X X 5 3 3 3 X 3 3 1 1 0 0 2 3 4 3 2 1 0 0 0
2 2 2 1 1 1 1 3 X X X X X 2 3 X 3 1 0 0 0 1 X 2 3 X 2 0 0 0
1 1 2 X 1 0 0 1 2 3 3 3 2 1 2 X X 1 0 0 0 1 1 2 X X 2 1 1 1
1 X 2 1 2 1 1 0 0 0 0 0 0 0 2 3 3 2 1 1 0 0 0 2 3 3 1 2 X 2
1 1 1 1 2 X 2 2 1 1 1 1 2 2 3 X 1 1 X 2 1 0 0 1 X 3 2 3 X 3
1 1 1 2 X 4 X 2 X 2 2 X 2 X X 2 1 1 2 X 1 0 0 1 3 X X 3 3 X
1 X 1 2 X 4 2 2 2 X 2 1 3 3 3 1 0 1 3 3 2 0 0 0 3 X X 2 2 X
1 1 1 1 2 X 1 0 1 1 1 0 1 X 2 1 2 2 X X 2 1 0 0 2 X 4 2 2 1
1 1 0 1 2 2 1 0 0 1 1 2 2 2 3 X 4 X 4 3 X 1 1 1 2 1 2 X 1 0
X 2 2 3 X 2 1 0 0 1 X 2 X 1 2 X 4 X 3 3 2 3 2 X 1 0 1 2 3 2
1 2 X X 3 X 1 0 0 1 1 2 1 1 1 1 2 2 X 2 X 2 X 2 1 0 0 1 X X
```

資料結構

我使用的資料結構圖示與說明如下：



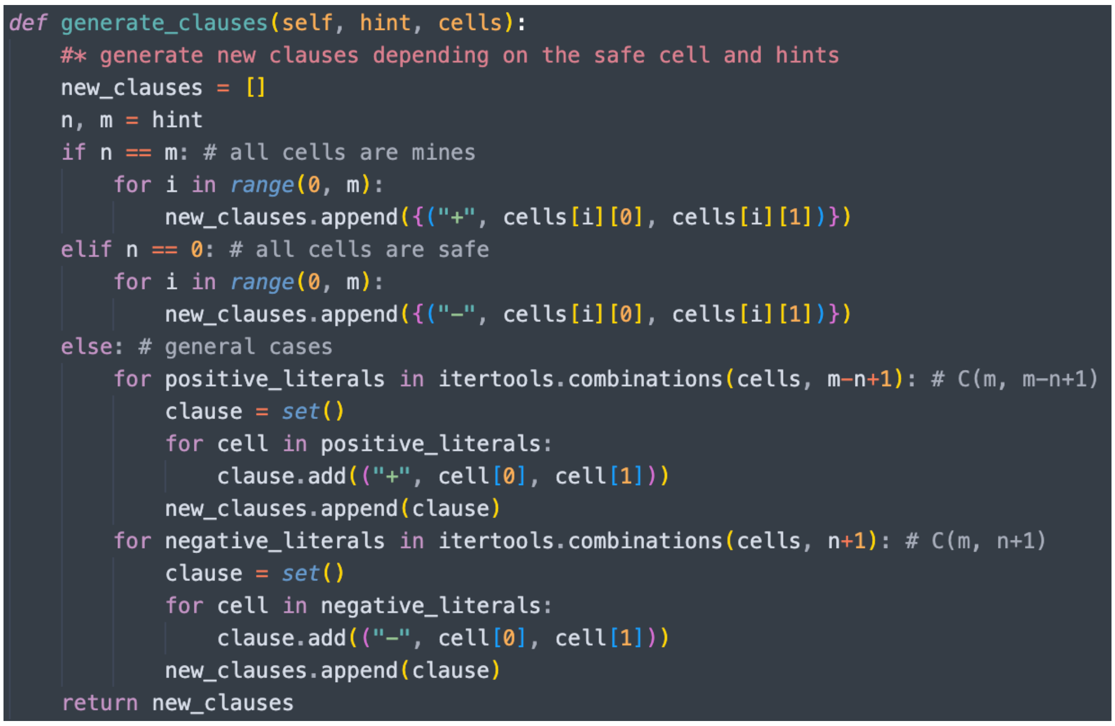
cell 資料結構：tuple，內含三個元素，第一個元素用 + 或 − 來代表 positive 或 negative，第二三個元素代表 cell 的座標

clause 資料結構：set，因為很常需要在 clause 裡面尋找特定的 cell 或比較子集，所以選用集合。裡面包含數個 cell，這是 CNF clauses，所以不同 cell 間是使用 or 相連

KB 資料結構：list，裡面包含所有 clauses

About generating clauses from the hints:

generate_clauses 分成三大不同的狀況產生相對應的 clauses 並回傳。

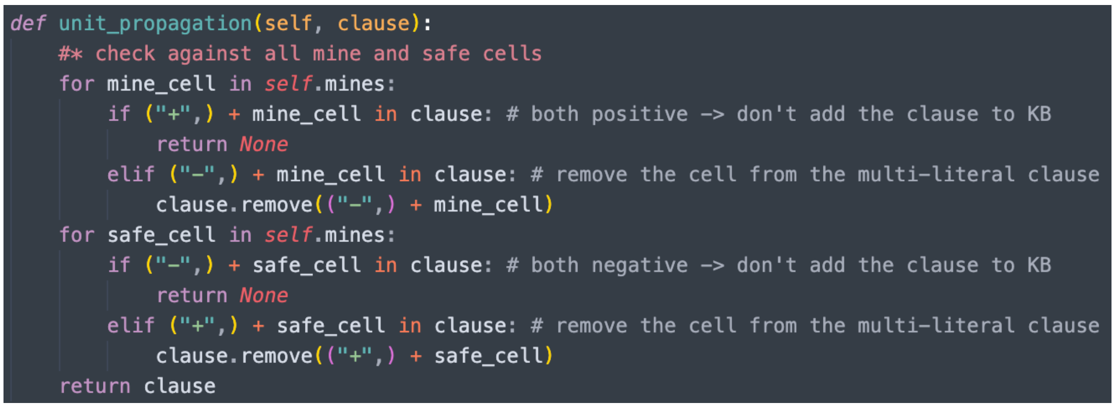


```python
def generate_clauses(self, hint, cells):
    #* generate new clauses depending on the safe cell and hints
    new_clauses = []
    n, m = hint
    if n == m: # all cells are mines
        for i in range(0, m):
            new_clauses.append({("+", cells[i][0], cells[i][1])})
    elif n == 0: # all cells are safe
        for i in range(0, m):
            new_clauses.append({("-", cells[i][0], cells[i][1])})
    else: # general cases
        for positive_literals in itertools.combinations(cells, m-n+1): # C(m, m-n+1)
            clause = set()
            for cell in positive_literals:
                clause.add(("+", cell[0], cell[1]))
            new_clauses.append(clause)
        for negative_literals in itertools.combinations(cells, n+1): # C(m, n+1)
            clause = set()
            for cell in negative_literals:
                clause.add(("-", cell[0], cell[1]))
            new_clauses.append(clause)
    return new_clauses
```

About inserting a new clause to the KB:

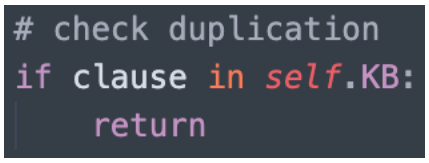
insert_clause()分為以下四個步驟:

1. unit propagation



```python
def unit_propagation(self, clause):
    #* check against all mine and safe cells
    for mine_cell in self.mines:
        if ("+",) + mine_cell in clause: # both positive -> don't add the clause to KB
            return None
        elif ("-",) + mine_cell in clause: # remove the cell from the multi-literal clause
            clause.remove(("-",) + mine_cell)
    for safe_cell in self.mines:
        if ("-",) + safe_cell in clause: # both negative -> don't add the clause to KB
            return None
        elif ("+",) + safe_cell in clause: # remove the cell from the multi-literal clause
            clause.remove(("+",) + safe_cell)
    return clause
```

2. check duplication



```python
# check duplication
if clause in self.KB:
    return
```

3. check subsumption

```python
# check subsumption
less_strict_list = []
insert_flag = 1
for old_clause in self.KB:
    if self.subsumption(clause, old_clause): # if new is subset of old
        less_strict_list.append(old_clause) # remove less strict one(old)
    if self.subsumption(old_clause, clause): # if old is subset of new
        insert_flag = 0 # skip insertion
for discard in less_strict_list: # remove all less strict elements
    try:
        self.KB.remove(discard)
    except:
        pass # duplication
```

4. 當通過以上三個檢查，才可以新增到 KB

```python
if insert_flag:
    self.KB.append(clause)
```

About "matching" two clauses:

matching()分為三個步驟：

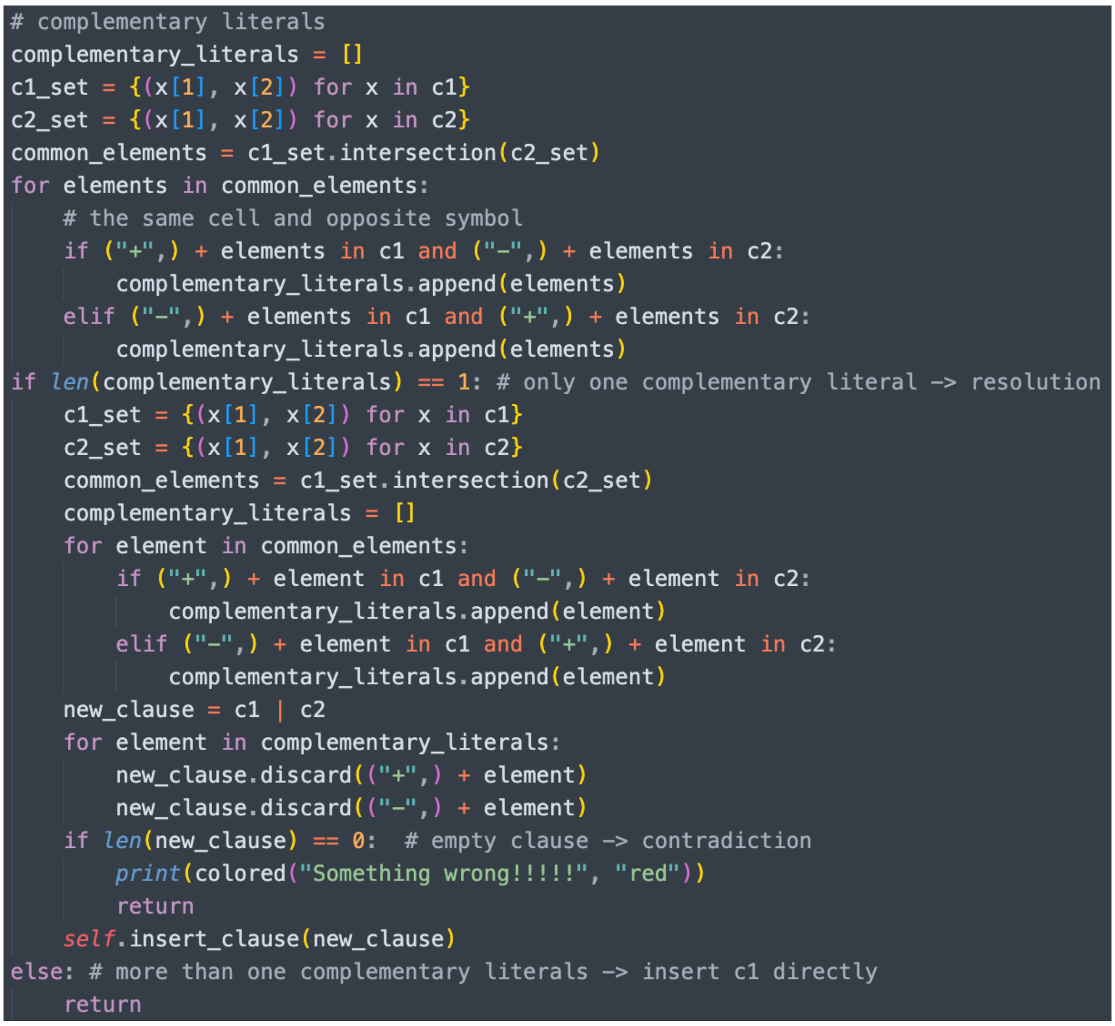
1. check duplication

```python
if c1 == c2:
    self.matching_remove_list.append(c1)
    return
```

2. check subsumption

```python
# check subsumption
if self.subsumption(c1, c2): # check whether c1 is subset of c2
    self.matching_remove_list.append(c2) # remove less strict one(c2)
    return
if self.subsumption(c2, c1): # check whether c2 is subset of c1
    self.matching_remove_list.append(c1) # remove less strict one(c1)
    return
```

3. complementary literal and resolution：先逐一檢查並找出所有 complementary literal（我的做法是先取出 clause 內 cell 座標的部分，取兩者的交集，然後再檢查同樣的 cell 是否有不同的符號），如果這兩個 clause 間只有一對，則進行 resolution，產生新的 clause 並檢查後插入，若超過一對則不進行 resolution（因為 resolution 會導致恆真）



```python
# complementary literals
complementary_literals = []
c1_set = {(x[1], x[2]) for x in c1}
c2_set = {(x[1], x[2]) for x in c2}
common_elements = c1_set.intersection(c2_set)
for elements in common_elements:
    # the same cell and opposite symbol
    if ("+",) + elements in c1 and ("-",) + elements in c2:
        complementary_literals.append(elements)
    elif ("-",) + elements in c1 and ("+",) + elements in c2:
        complementary_literals.append(elements)
if len(complementary_literals) == 1: # only one complementary literal -> resolution
    c1_set = {(x[1], x[2]) for x in c1}
    c2_set = {(x[1], x[2]) for x in c2}
    common_elements = c1_set.intersection(c2_set)
    complementary_literals = []
    for element in common_elements:
        if ("+",) + element in c1 and ("-",) + element in c2:
            complementary_literals.append(element)
        elif ("-",) + element in c1 and ("+",) + element in c2:
            complementary_literals.append(element)
    new_clause = c1 | c2
    for element in complementary_literals:
        new_clause.discard(("+",) + element)
        new_clause.discard(("-",) + element)
    if len(new_clause) == 0:  # empty clause -> contradiction
        print(colored("Something wrong!!!!!", "red"))
        return
    self.insert_clause(new_clause)
else: # more than one complementary literals -> insert c1 directly
    return
```

結果展示與實驗

the effect of global constraints

Medium（16x16 有 25 個地雷）initial steps = sqrt(#cells) / 2

global constraints 加入時機	成功機率（隨機跑 20 場）
不進行	85%
剩 9 個 unmarked 時	90%
剩 10%unmarked 時（約 25 個）	約 30%會運行過久（超過三分鐘）

若在後期加入 global constraints 可以稍微提高成功的機率，但效果沒有想像中顯著。原本的想法是使用 unmarked 比例來當進入後期的依據，但發現這樣會造成 unmarked cell 太多而造成新產生的 clause 太多，造成運行時間過長，因為正常狀況 hints 的 unmarked cell 會在 9 以內，所以最後決定用剩下 9 個 unmarked cell 當作加入 global constraints 的時機。

changing the number of initial safe cells

Medium（16x16 有 25 個地雷）剩 9 個 unmarked 做 global constraints

initial steps	成功機率（隨機跑 20 場）
sqrt(#cells) * 2	100%
sqrt(#cells)	90%
sqrt(#cells) / 2	95% 有些在早期就 stuck
sqrt(#cells) / 4	90% 有些在早期就 stuck

其實差距並不大，成功率都很高，但 initial safe cells 越小越有可能因為在前期拿不到有用資訊而很早就 stuck

Hard（16x30 有 99 個地雷）剩 9 個 unmarked 做 global constraints

initial steps	成功機率（隨機跑 20 場）
sqrt(#cells) * 2	40%
sqrt(#cells)	35%
sqrt(#cells) / 2	15%
sqrt(#cells) / 4	0%

因為難度提高，若一開始的資訊量太少，基本上就很難成功。

遇到的困難與心得

KB 無法收斂

在實作時有遇到 KB 遲遲無法收斂的問題，經過不斷的 trace code 才慢慢把整個邏輯變得更加完善，其中我發現插入前的檢查非常重要，無論是什麼情況下產生的新 clause，都必須要先做 unit propagation 和 check duplication 和 subsumption 才能加入 KB，不然就會難以收斂。
另外也要避免一輪裡產生太多新的 clause，例如我一開始太早加入 global constraint 的條件，導致運行量太大而無法收斂 KB。

self.KB.remove()的時機

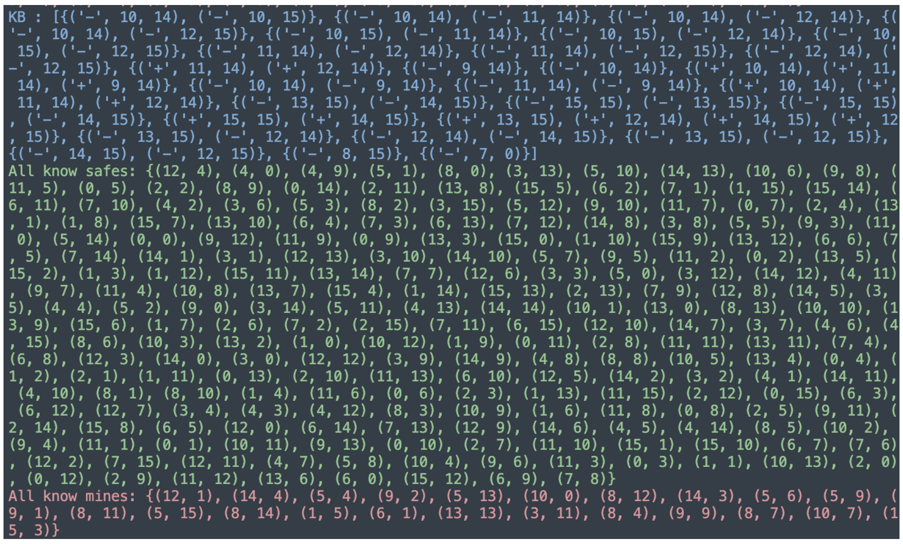
實作時也發現移除 KB 中 clause 的時機很重要，因為若在迭代 KB 時做移除，將會導致列表的長度跟元素位置改變，有些元素可能會被重複訪問或沒被訪問到，也會造成 KB 無法收斂，所以後來我都是先儲存起要刪除的 clause，在迭代 KB 完成後再做刪除。

資料結構的選擇

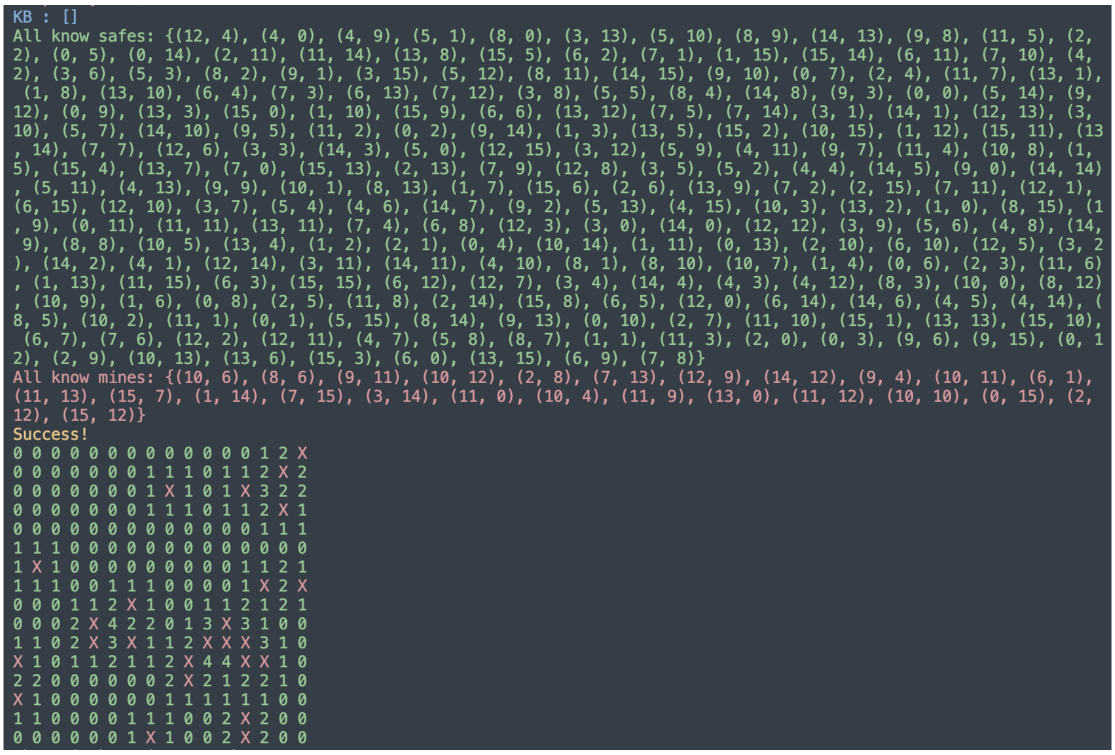
一開始我都使用 list 作為容器，後來發現這樣查找並不方便，最後 clause 才選擇使用 set，增加程式運行的效率。

遊戲介面

本來有想要使用圖形化介面，後來發現使用 termcolor 的 colored 就可以很清楚的展現程式的結果。



```
KB : [{('-', 10, 14), ('-', 10, 15)}, {('-', 10, 14), ('-', 11, 14)}, {('-', 10, 14), ('-', 12, 14)}, {(
'-', 10, 14), ('-', 12, 15)}, {('-', 10, 15), ('-', 11, 14)}, {('-', 10, 15), ('-', 12, 14)}, {('-', 10,
 15), ('-', 12, 15)}, {('-', 11, 14), ('-', 12, 14)}, {('-', 11, 14), ('-', 12, 15)}, {('-', 12, 14), ('
-', 12, 15)}, {('+', 11, 14), ('+', 12, 14)}, {('-', 9, 14)}, {('-', 10, 14)}, {('+', 10, 14), ('+', 11,
 14), ('+', 9, 14)}, {('-', 10, 14), ('-', 9, 14)}, {('-', 11, 14), ('-', 9, 14)}, {('+', 10, 14), ('+',
 11, 14), ('+', 12, 14)}, {('-', 13, 15), ('-', 14, 15)}, {('-', 15, 15), ('-', 13, 15)}, {('-', 15, 15)
, ('-', 14, 15)}, {('+', 15, 15), ('+', 14, 15)}, {('+', 13, 15), ('+', 12, 14), ('+', 14, 15), ('+', 12
, 15)}, {('-', 13, 15), ('-', 12, 14)}, {('-', 12, 14), ('-', 14, 15)}, {('-', 13, 15), ('-', 12, 15)},
{('-', 14, 15), ('-', 12, 15)}, {('-', 8, 15)}, {('-', 7, 0)}]
All know safes: {(12, 4), (4, 0), (4, 9), (5, 1), (8, 0), (3, 13), (5, 10), (14, 13), (10, 6), (9, 8), (
11, 5), (0, 5), (2, 2), (8, 9), (0, 14), (2, 11), (13, 8), (15, 5), (6, 2), (7, 1), (1, 15), (15, 14), (
6, 11), (7, 10), (4, 2), (3, 6), (5, 3), (8, 2), (3, 15), (5, 12), (9, 10), (11, 7), (0, 7), (2, 4), (13
, 1), (1, 8), (15, 7), (13, 10), (6, 4), (7, 3), (6, 13), (7, 12), (14, 8), (3, 8), (5, 5), (9, 3), (11,
 0), (5, 14), (0, 0), (9, 12), (11, 9), (0, 9), (13, 3), (15, 0), (1, 10), (15, 9), (13, 12), (6, 6), (7
, 5), (7, 14), (14, 1), (3, 1), (12, 13), (3, 10), (14, 10), (5, 7), (9, 5), (11, 2), (0, 2), (13, 5), (
15, 2), (1, 3), (1, 12), (15, 11), (13, 14), (7, 7), (12, 6), (3, 3), (5, 0), (3, 12), (14, 12), (4, 11)
, (9, 7), (11, 4), (10, 8), (13, 7), (15, 4), (1, 14), (15, 13), (2, 13), (7, 9), (12, 8), (14, 5), (3,
5), (4, 4), (5, 2), (9, 0), (3, 14), (5, 11), (4, 13), (14, 14), (10, 1), (13, 0), (8, 13), (10, 10), (1
3, 9), (15, 6), (1, 7), (2, 6), (7, 2), (2, 15), (7, 11), (6, 15), (12, 10), (14, 7), (3, 7), (4, 6), (4
, 15), (8, 6), (10, 3), (13, 2), (1, 0), (10, 12), (1, 9), (0, 11), (2, 8), (11, 11), (13, 11), (7, 4),
(6, 8), (12, 3), (14, 0), (3, 0), (12, 12), (3, 9), (14, 9), (4, 8), (8, 8), (10, 5), (13, 4), (0, 4), (
1, 2), (2, 1), (1, 11), (0, 13), (2, 10), (11, 13), (6, 10), (12, 5), (14, 2), (3, 2), (4, 1), (14, 11),
 (4, 10), (8, 1), (8, 10), (1, 4), (11, 6), (0, 6), (2, 3), (1, 13), (11, 15), (2, 12), (0, 15), (6, 3),
 (6, 12), (12, 7), (3, 4), (4, 3), (4, 12), (8, 3), (10, 9), (1, 6), (11, 8), (0, 8), (2, 5), (9, 11), (
2, 14), (15, 8), (6, 5), (12, 0), (6, 14), (7, 13), (12, 9), (14, 6), (4, 5), (4, 14), (8, 5), (10, 2),
(9, 4), (11, 1), (0, 1), (10, 11), (9, 13), (0, 10), (2, 7), (11, 10), (15, 1), (15, 10), (6, 7), (7, 6)
, (12, 2), (7, 15), (12, 11), (4, 7), (5, 8), (10, 4), (9, 6), (11, 3), (0, 3), (1, 1), (10, 13), (2, 0)
, (0, 12), (2, 9), (11, 12), (13, 6), (6, 0), (15, 12), (6, 9), (7, 8)}
All know mines: {(12, 1), (14, 4), (5, 4), (9, 2), (5, 13), (10, 0), (8, 12), (14, 3), (5, 6), (5, 9), (
9, 1), (8, 11), (5, 15), (8, 14), (1, 5), (6, 1), (13, 13), (3, 11), (8, 4), (9, 9), (8, 7), (10, 7), (1
5, 3)}
```



```
KB : []
All know safes: {(12, 4), (4, 0), (4, 9), (5, 1), (8, 0), (3, 13), (5, 10), (8, 9), (14, 13), (9, 8), (11, 5), (2,
2), (0, 5), (0, 14), (2, 11), (11, 14), (13, 8), (15, 5), (6, 2), (7, 1), (1, 15), (15, 14), (6, 11), (7, 10), (4,
2), (3, 6), (5, 3), (8, 2), (9, 1), (3, 15), (5, 12), (8, 11), (14, 15), (9, 10), (0, 7), (2, 4), (11, 7), (13, 1),
 (1, 8), (13, 10), (6, 4), (7, 3), (6, 13), (7, 12), (3, 8), (5, 5), (8, 4), (14, 8), (9, 3), (0, 0), (5, 14), (9,
12), (0, 9), (13, 3), (15, 0), (1, 10), (15, 9), (6, 6), (13, 12), (7, 5), (7, 14), (3, 1), (14, 1), (12, 13), (3,
10), (5, 7), (14, 10), (9, 5), (11, 2), (0, 2), (9, 14), (1, 3), (13, 5), (15, 2), (10, 15), (1, 12), (15, 11), (13
, 14), (7, 7), (12, 6), (3, 3), (14, 3), (5, 0), (12, 15), (3, 12), (5, 9), (4, 11), (9, 7), (11, 4), (10, 8), (1,
5), (15, 4), (13, 7), (7, 0), (15, 13), (2, 13), (7, 9), (12, 8), (3, 5), (5, 2), (4, 4), (14, 5), (9, 0), (14, 14)
, (5, 11), (4, 13), (9, 9), (10, 1), (8, 13), (1, 7), (15, 6), (2, 6), (13, 9), (7, 2), (2, 15), (7, 11), (12, 1),
(6, 15), (12, 10), (3, 7), (5, 4), (4, 6), (14, 7), (9, 2), (5, 13), (4, 15), (10, 3), (13, 2), (1, 0), (8, 15), (1
, 9), (0, 11), (11, 11), (13, 11), (7, 4), (6, 8), (12, 3), (3, 0), (14, 0), (12, 12), (3, 9), (5, 6), (4, 8), (14,
 9), (8, 8), (10, 5), (13, 4), (1, 2), (2, 1), (0, 4), (10, 14), (1, 11), (0, 13), (2, 10), (6, 10), (12, 5), (3, 2
), (14, 2), (4, 1), (12, 14), (3, 11), (14, 11), (4, 10), (8, 1), (8, 10), (10, 7), (1, 4), (0, 6), (2, 3), (11, 6)
, (1, 13), (11, 15), (6, 3), (15, 15), (6, 12), (12, 7), (3, 4), (14, 4), (4, 3), (4, 12), (8, 3), (10, 0), (8, 12)
, (10, 9), (1, 6), (0, 8), (2, 5), (11, 8), (2, 14), (15, 8), (6, 5), (12, 0), (6, 14), (14, 6), (4, 5), (4, 14), (
8, 5), (10, 2), (11, 1), (0, 1), (5, 15), (8, 14), (9, 13), (0, 10), (2, 7), (11, 10), (15, 1), (13, 13), (15, 10),
 (6, 7), (7, 6), (12, 2), (12, 11), (4, 7), (5, 8), (8, 7), (1, 1), (11, 3), (2, 0), (0, 3), (9, 6), (9, 15), (0, 1
2), (2, 9), (10, 13), (13, 6), (15, 3), (6, 0), (13, 15), (6, 9), (7, 8)}
All know mines: {(10, 6), (8, 6), (9, 11), (10, 12), (2, 8), (7, 13), (12, 9), (14, 12), (9, 4), (10, 11), (6, 1),
(11, 13), (15, 7), (1, 14), (7, 15), (3, 14), (11, 0), (10, 4), (11, 9), (13, 0), (11, 12), (10, 10), (0, 15), (2,
12), (15, 12)}
Success!
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 X
0 0 0 0 0 0 0 1 1 1 0 1 1 2 X 2
0 0 0 0 0 0 0 1 X 1 0 1 X 3 2 2
0 0 0 0 0 0 0 1 1 1 0 1 1 2 X 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 X 1 0 0 0 0 0 0 0 0 0 1 1 2 1
1 1 1 0 0 1 1 1 0 0 0 0 1 X 2 X
0 0 0 1 1 2 X 1 0 0 1 1 2 1 2 1
0 0 0 2 X 4 2 2 0 1 3 X 3 1 0 0
1 1 0 2 X 3 X 1 1 2 X X X 3 1 0
X 1 0 1 1 2 1 1 2 X 4 4 X X 1 0
2 2 0 0 0 0 0 0 2 X 2 1 2 2 1 0
X 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0
1 1 0 0 0 0 1 1 1 0 0 2 X 2 0 0
0 0 0 0 0 0 1 X 1 0 0 2 X 2 0 0
```

Code Appendix (github)

https://github.com/moirachen1019/Artificial-Intelligence-Capstone/blob/main/HW3/minesweeper.py