## 初始設定

1. 設定空的圈圈叉叉的圖以及 AI 和 real player 所代表的符號

```
def __init__(self):
    self.board = [
        ['', '', ''],
        ['', '', ''],
        ['', '', '']
    ]

    self.ai = 'O'
    self.realPlayer = 'X'

    # 印出現在的圈圈叉叉圖
    self.print_broad(0)
```

2. current_player_minimax function

```
def current_player_minimax(self, round_number, player):
    # 判斷現在是誰的局
    if player == 'O':   # ai
        self.best_move()

    elif player == 'X':     # real player
        correct = False
        while not correct:
            input_number = input('Your move: ')
            if input_number == '0':
                return True
            print()
            row, col = self.number_to_rowcol(int(input_number))
            if row < 0 or col < 0:
                correct = False
            else:
                if self.board[row][col] == '':
                    correct = True
                    self.board[row][col] = self.realPlayer

                else:
                    correct = False

    # 印出現在的圈圈叉叉圖
    self.print_broad(round_number)
    return False
```

如果傳入的符號為 O：為 AI 可以移動的局，使用 minimax 演算法
如果傳入的符號為 X：為 real player 可以移動的局，請輸入數字
若使用者輸入的數字為 0，即為退出遊戲

3. best_move function

```
def best_move(self):
    best_score = -1
    best_move = [-1, -1]
    for i in range(3):
        for j in range(3):
            if self.board[i][j] == '':
                self.board[i][j] = self.ai  # 先將那格設給ai
                score = self.minimax(0, False)   # depth = 0, 下一個player是min player(real player)
                self.board[i][j] = ''   # 還原剛剛的假設
                if score > best_score:
                    best_score = score
                    best_move = [i, j]

    # 選出最好的移動後 再移動
    self.board[best_move[0]][best_move[1]] = self.ai
    move_number = self.rowcol_to_number(best_move[0], best_move[1])
    print("AI's move:", move_number)
    print()
```

用 minimax 演算法找到最佳移動的位置

4. minimax function

```
def minimax(self, depth, isMaxPlayer):
    winner, game_over = self.check_winner()
    if game_over:
        if winner == self.ai:
            return 1
        elif winner == self.realPlayer:
            return -1
        else:   # tie
            return 0

    best_score = 0
    if isMaxPlayer:     # max player is ai
        for i in range(3):
            for j in range(3):
                if self.board[i][j] == '':
                    self.board[i][j] = self.ai
                    score = self.minimax(depth+1, False)
                    self.board[i][j] = ''
                    best_score = max(score, best_score)

        return best_score

    else:       # real player is min player
        for i in range(3):
            for j in range(3):
                if self.board[i][j] == '':
                    self.board[i][j] = self.realPlayer
                    score = self.minimax(depth+1, True)
                    self.board[i][j] = ''
                    best_score = min(score, best_score)

        return best_score
```

設 AI 為 maximizing player，real player 為 minimizing player
若格子為空(尚未被走過)，就去試試是否為好的走法
(1) 若為 maximizing player，好的走法為最大的 score
(2) 若為 minimizing player，好的走法為最小的 score
(3) 若為可以分出勝負和平手的情況，則直接回傳分數
AI win = 1、real player win = -1、tie = 0

5. number_to_rowcol function and rowcol_to_number function
這兩個 function 為輸入執行和顯示的轉換使用

```python
def number_to_rowcol(self, input_number):
    if 0 < input_number <= 9:
        if input_number == 1:
            return 0, 0
        elif input_number == 2:
            return 0, 1
        elif input_number == 3:
            return 0, 2
        elif input_number == 4:
            return 1, 0
        elif input_number == 5:
            return 1, 1
        elif input_number == 6:
            return 1, 2
        elif input_number == 7:
            return 2, 0
        elif input_number == 8:
            return 2, 1
        elif input_number == 9:
            return 2, 2

    else:
        return -1, -1
```

```python
def rowcol_to_number(self, row, col):
    if row == 0 and col == 0:
        return 1
    elif row == 0 and col == 1:
        return 2
    if row == 0 and col == 2:
        return 3
    elif row == 1 and col == 0:
        return 4
    if row == 1 and col == 1:
        return 5
    elif row == 1 and col == 2:
        return 6
    if row == 2 and col == 0:
        return 7
    elif row == 2 and col == 1:
        return 8
    if row == 2 and col == 2:
        return 9
    else:
        return -1
```

6. print_board function
用來顯示圈圈叉叉的圖

```python
def print_broad(self, round_number):  # 印出圈圈叉叉的圖
    print('Round', round_number, ':')
    number = 0
    for i in range(len(self.board)):
        print('|', end='')
        for j in range(len(self.board[i])):
            number += 1
            if self.board[i][j] == '':
                print(number, end='|')
            else:
                print(self.board[i][j], end='|')
        print()
```

其中若該格尚未被填過，則顯示數字(1~9)，反之則填 O 或 X

7. check_winner function

用來判斷是否有連線

```python
def check_winner(self):    # 判斷是否有連線、誰是winner
    # horizontal
    for i in range(len(self.board)):
        if self.board[i][0] == self.board[i][1] == self.board[i][2] and self.board[i][0] != '':
            return self.board[i][0], True

    # vertical
    for i in range(len(self.board[0])):
        if self.board[0][i] == self.board[1][i] == self.board[2][i] and self.board[0][i] != '':
            return self.board[0][i], True

    # diagonal
    if self.board[0][0] == self.board[1][1] == self.board[2][2] and self.board[0][0] != '':
        return self.board[0][0], True

    if self.board[0][2] == self.board[1][1] == self.board[2][0] and self.board[0][2] != '':
        return self.board[0][2], True

    # if full -> tie
    count = 0
    for i in range(len(self.board)):
        for j in range(len(self.board[i])):
            if self.board[i][j] != '':
                count += 1

    if count == 9:
        return 'tie', True

    return None, False
```

(1) 若有連線即為有贏家出現，回傳贏家是誰和結束遊戲(True)

(2) 如果全部格子皆被填滿且沒有連線，回傳平手和結束遊戲(True)

(3) 若上述皆無，則遊戲繼續，回傳 None 和繼續遊戲(False)

## 程式執行

```
game_over = False
game_number = 1
while not game_over:
    if game_number % 2 == 1:
        now_player = 'X'    # real player
    else:
        now_player = 'O'    # ai

    break_game = self.current_player_minimax(game_number, now_player)
    if break_game:
        print('Break the game')
        break

    winner, game_over = self.check_winner()
    if game_over:
        if winner == 'O':
            print('AI is winner!')
        elif winner == 'X':
            print('You are winner!')
        else:
            print('The game is a tie!')

    game_number += 1
```

1. 判斷現在是誰移動(奇數為 real player，偶數為 AI)
2. 若使用者輸入 0，即為退出遊戲
3. 判斷是否有連線或平手，並印出結果

## 執行結果

```
Round 0 :
|1|2|3|
|4|5|6|
|7|8|9|
Your move: 5

Round 1 :
|1|2|3|
|4|X|6|
|7|8|9|
AI's move: 1

Round 2 :
|O|2|3|
|4|X|6|
|7|8|9|
Your move: 7

Round 3 :
|O|2|3|
|4|X|6|
|X|8|9|
AI's move: 3
```

```
Round 4 :
|O|2|O|
|4|X|6|
|X|8|9|
Your move: 2

Round 5 :
|O|X|O|
|4|X|6|
|X|8|9|
AI's move: 8

Round 6 :
|O|X|O|
|4|X|6|
|X|O|9|
Your move: 6

Round 7 :
|O|X|O|
|4|X|X|
|X|O|9|
AI's move: 4
```

```
Round 8 :
|O|X|O|
|O|X|X|
|X|O|9|
Your move: 9

Round 9 :
|O|X|O|
|O|X|X|
|X|O|X|
The game is a tie!
```