1.

首先嘗試純數學運算!! 這裡計算未消除一次性機率 即卡牌不消除 計算消除理論上變成無限可能 無法嘗試用數學計算

檔案名稱 : slot math.py

以下將原始程式碼由藍字表示 用紅字表示對程式碼的解釋

```
# 純數學機率運算  只能算為純消除前的機率期望值!!   有# 之後視為附錄  程式不讀取
定義函數 排列組合Cn去m  又n個樣品取m個排列
def c(n,m): #排列組合C n 取 m
    def p(x):
        a = 1
        for s in range(1,x+1):
            a = a*s
        return a
    return p(n)//(p(m)*p(n-m))
回傳計算值
首先計算沒有W的
 #沒有W在連貫中
3連 排列可能3種 : NNAAA, NAAAN, AAANN     N代表非相同於A,W的數字
con3 = ((1*1*1*8*8)/100000)*3*106  # 排列方式:3 三連貫分數加總:106
3連 W不在連續中 排列可能2種 : WNAAA, NAAAW
con3w = ((1*1*1*8*1)/100000)*2*106  # 排列方式:2 AAANW WNAAA
4連 排列可能2種 : NAAAA, AAAAN.
con4 = ((1*1*1*1*8)/100000)*2*427  # 排列方式:2  四連貫分數加總:427
5連 排列可能1種 : AAAAA, AAAAN.
con5 = ((1*1*1*1*1)/100000)*1*4525  # 排列方式:1  五連貫分數加總:4525

計算有W的情況
 #有W在連貫中
3連 W在連續中 排列可能3種 :  NN(AAW). N(AAW)N. (AAW)NN
(AAW) 判讀為3個A  內含1或2個W  總體組合方式(c(3,1)+c(3,2))
wcon3 = ((1*1*1*8*8)/100000)*3*106*(c(3,1)+c(3,2)) # 三連貫有1和2個W
3連 W在連續中 也在剩餘兩格中 排列可能2種 :  WN(AAW), (AAW)NW
(AAW) 判讀為3個A  內含1或2個W  總體組合方式(c(3,1)+c(3,2))
wcon3w = ((1*1*1*8*1)/100000)*2*106*(c(3,1)+c(3,2)-2) # (WWA)BW WB(WWA)
2個3連 W在連續中 排列可能2種 :  AAWBB ,WBWAW
w2con3 = ((9*1*1*8*1)/100000)*2*23.556 # WAWBW 平均雙三連貫分數加總:23.556
1個3連 1個4連 排列可能2種 :  AWWBW ,WBWWA
w2con34 = ((9*1*1*8*1)/100000)*2*59.222 # AWWBW 平均三連貫四連貫分數加總:
59.222
4連 W在連續中 排列可能2種 :  N(AAAW), (AAAW)N
(AAAW) 判讀為4個A  內含1~3個W  總體組合方式(c(4,1)+c(4,2)+c(4,3))
wcon4 = ((1*1*1*1*8)/100000)*2*427*(c(4,1)+c(4,2)+c(4,3)-5) # 四連貫有1.2.3個W

2個4連  , W在連續中  排列可能2種 :  AAWWB ,AWWBW
```

w2con4 = ((9*1*1*1*8)/100000)*2*94.889 # WAWWB 平均雙四連貫分數加總:
94.889*2
5連 排列可能1種 : (AAAAW)
(AAAAW) 判讀為4個A 內含1~4個W 總體組合方式(c(5,1)+c(5,2)+c(5,3)+c(5,4))
wcon5 = ((1*1*1*1*1)/100000)*1*4525*(c(5,1)+c(5,2)+c(5,3)+c(5,4)) # 五連貫有
1.2.3.4個W
期望值相加
exp_val =
(con3+con3w+con4+con5+wcon3+wcon3w+w2con3+w2con34+wcon4+w2con4+wco
n5)
print('RTP:%f%%'%(exp_val*100))

Ans: exp_val = 385.123048%

2.
以上數學計算複雜, 且容易沒有考慮到某種情況就會錯誤, 如上所說這種計算方式也無法作為消除後的情況假設(無限種算法)

所以使用Python 寫一台符合題目的原型機 利用模擬遊玩多次取得平均的RTP值!!
先寫出所需要的函數~
檔案名稱 : slot_func.py

import numpy as np 導入python 模組Numpy 和 Pandas
import pandas as pd
cards = 'WABCDEFGHI' # 卡牌可能 將卡牌可能放進一個str序列中 方便取用

將賠付線設定為一排5個的數列 ex: 賠付線1 : 2, 2, 2, 2, 2 (都是中間那格2)
賠付線4 : 1, 2, 3, 2, 1
寫入一個excel檔並儲存csv在讀入此表格, 讀取路線時使用此表格
line_way = pd.read_csv('line_way.csv') # 賠付線

將卡牌賠率表寫入一個excel檔並儲存csv,在讀入此表格, 算分時讀取此表格
card_score = pd.read_csv('card_score.csv').set_index('pay_table') # 卡牌賠率表

將卡牌所有的座標寫入一個集合 3x5 即15種可能!!
crdset = {(1,1),(1,2),(1,3),(1,4),(1,5),(2,1),(2,2),(2,3),(2,4),(2,5), # 所有座標
(3,1),(3,2),(3,3),(3,4),(3,5)}

開始定義第一個函數 產生一個3x5 並有10種可能 'WABCDEFGHI'
出現1 代表W 2代表A 依此類推 使用函式會輸出類似以下的表格Array
array([[1, 6, 10, 9, 9],
[7, 7, 7, 2, 1],
[5, 2, 8, 2, 6]])

```
def spin(): #產生一組 3X5 隨機卡牌組 column可重複!!  p:調整每張牌出現機率!!  輸出數字方便運算   使用隨機模組random.choice
    arr = np.random.choice(
            [1,2,3,4,5,6,7,8,9,10], 15
            , p=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,]
            ).reshape(3, 5) # 隨機抽取 3x5 array 可重複
    return arr

為了方便判讀  這是將數字轉為卡牌的函式  輸入以上的排列會輸出以下:
array([['W', 'E', 'I', 'H', 'H'],
            ['F', 'F', 'F', 'A', 'W'],
            ['D', 'A', 'G', 'A', 'E']], dtype='<U1')
def numtocrd_arr(numarr): # 數字(int)array 轉 Cards 字母(str)array
    arr = np.full((3, 5),'')
    for n in range(3):
        for m in range(5):
            arr[n][m] = cards[numarr[n, m]-1]
    return arr

讀取賠付線  輸入spin( ) 輸出的Array  和要讀取的賠付線
這裡line(arr, 2)  這裡舉例讀取第二條賠付線(最上一條線)
輸出是一個序列  ['W', 'E', 'I', 'H', 'H']
def line(arr, num): # 讀取賠付線 line (3x5卡牌array, line_number(1~20))
    x_ind = line_way.iloc[num-1]
    arr = numtocrd_arr(arr)
    lst = []
    for n in range(5):
        lst.append(arr[(x_ind[n]-1), n])
    return lst

為了判讀每條線的分數, 首先要知道卡牌的連續性, 連續性跟有沒有W有很大的關係, 這個函式先討論沒有W的情況, 另外從卡牌賠率表得知, 只有3個相連才會有分數, 只需要確定這條賠付線中是否有超過3個相連的狀況出現
方法式和旁邊的卡牌是否相同, 相同就輸出卡牌  不相同輸出N
ex : AABBD
A&A 相同 輸出A
A&B 不相同  輸出N
B&B 相同 輸出B
B&D 不相同  輸出N
輸出為ANBN  若將這函式 使用兩次 就能得到我們所需要的資料:
沒有連線  會輸出NNN
輸出有一個A 代表A 3連
2個A 代表A 4連
3個A 代表A 5連
def same_test(lst_a): # 計算沒有 W line的連貫數 func(func( )) 就能獲得足夠資訊判斷連貫數
```

```
    count = len(lst_a) # AABBD -> ANBN -> NNN   AAABB -> AANB -> ANN 1A 3連貫
    lst_b = []  # AAAAB -> AAAN -> AAN 2A 4連貫   AAAAA -> AAAA -> AAA 3A 5連
貫
    for n in range(count-1): # 鄰近相同輸出卡牌編號 不同輸出N
        if lst_a[n] == lst_a[n+1]:
            lst_b.append(lst_a[n])
        else:
            lst_b.append('N')
    return lst_b
```

另外一個情況是有W, W可以當作任何卡牌, 這裡只要周圍是W 就輸出原來的卡牌
ex : AWBCW　這裡會輸出ABNC
A&W 遇到W 輸出A
W&B 遇到W　輸出B
B&C 不相同 輸出N
C&W 遇到W 輸出C
若是W遇到W　一樣輸出W　不衝突　最後兩次代入輸出的是3個值 W 可視為周圍相同的卡牌, 來計算連貫數, 以下判讀會在解釋!!

```
def same_test_W_in(lst_a): # 計算有 W line的連貫數 func(func( )) 就能獲得足夠資訊
判斷連貫數
    count = len(lst_a) # WABWD -> ANBD -> NNN   WAABW -> AANB -> ANN A 3連
貫 AAWBB AB雙3連貫
    lst_b = [] # AWWWB -> AWWB -> AWB AB 雙4連貫 AWWBW -> AWBB -> ABB A
3連貫 B 4連貫
    for n in range(count-1): # 鄰近相同輸出卡牌編號 不同輸出N
        if lst_a[n] == 'W':
            lst_b.append(lst_a[n+1])
        elif lst_a[n+1] == 'W':
            lst_b.append(lst_a[n])
        elif lst_a[n] == lst_a[n+1]:
            lst_b.append(lst_a[n])
        else:
            lst_b.append('N')
    return lst_b
```

將以上兩個判讀方式放進這個算分數的函式: 輸入是我們上方 line(arr, num) 函式輸出的某條賠付線, 其中card 為連線的卡牌編號A~I　get_num 定義為連線數3~5
之後計算分數將 card 和 get_num 代入卡牌賠率表算取分數, 因為一條賠付線最多會有兩條連線 所以 card, get_num, card2, get_num2 共兩組
這裡值得一提的是, 需要寫出所有可能的判斷數, 另外若沒有連線則輸出 W,1 連線, 分數為0分.

```
def line_score(line): # 計算分數 輸出分數 和 連貫資料
    if 'W' not in line:  # 沒有 W 情況
        lst = same_test(same_test(line)) # 用無W函式 same_test(lst_a)
        if lst.count('N') == 3: # 輸出3個N  代表沒有連線  輸出W, 1, W, 1
            card, get_num, card2, get_num2 = 'W', 1, 'W', 1
```

```
        elif lst.count('N') == 2: # 有2個N  代表有某卡片3連線
            same_card = (set(lst) & set(cards)).pop() #和卡片集合做交集, 取得卡片值
            card, get_num, card2, get_num2 =  same_card, 3, 'W', 1

        elif lst.count('N') == 1: # 有1個N  代表有某卡片4連線
            same_card = (set(lst) & set(cards)).pop() #和卡片集合做交集, 取得卡片值
            card, get_num, card2, get_num2 =  same_card, 4, 'W', 1

        else: # 無 W 5連貫 # 沒有N  代表有某卡片4連線
            same_card = (set(lst) & set(cards)).pop() #和卡片集合做交集, 取得卡片值
            card, get_num, card2, get_num2 =  same_card, 5, 'W', 1

    else: # 1~5個 W 有 W 情況
        lst = same_test_W_in(same_test_W_in(line)) # 用有W函 same_test_W_in(lst_a)
        if lst.count('N') == 3: # 輸出3個N  代表沒有連線  輸出W, 1, W, 1
            card, get_num, card2, get_num2 = 'W', 1, 'W', 1

        elif lst.count('N') == 2: # 有2個N  代表有某卡片3連線
            same_card = (set(lst) & set(cards)).pop() #和卡片集合做交集, 取得卡片值
            card, get_num, card2, get_num2 =  same_card, 3, 'W', 1

        elif lst.count('N') == 1:  # 有1個N  代表:
4連貫(AAN, AWN, WAN)
或雙3連貫(ANB) 原本排列 : AAWBB
            if len(set(lst)-{'N', 'W'}) == 1: # 扣掉N和W 之後剩下一個元素代表4連貫
                same_card = list(set(lst)-{'N', 'W'}).pop()
                card, get_num, card2, get_num2 =  same_card, 4, 'W', 1
            else: # 其餘可能雙3連貫
                set_pop = set(lst) & set(cards)
                same_card = set_pop.pop()
                same_card2 = set_pop.pop()
                card, get_num, card2, get_num2 =  same_card, 3, same_card2, 3
        else: # 剩下的可能 :  沒有N情況
W5連貫(WWW)
非W的5連貫(AAA, AAW, AWA, WAA, WWA,  AWW)
雙4連貫(AWB)  原本排列 : AWWWB
3連貫+4連貫 (ABB, BBA)  原本排列 : AWWBB, BBWWA
            set_a = set(lst)
            set_a.discard('W') # 扣掉W
            if len(set_a) == 0: # 之後空集代表W5連貫
                card, get_num, card2, get_num2 =  'W', 5, 'W', 1
            elif len(set_a) == 1: #  之後剩下一個元素代表非W卡片5連貫
                same_card = (set(lst) & set(cards)).pop()
                card, get_num, card2, get_num2 =  same_card, 5, 'W', 1
            elif len(set(lst)) == 3: # 重新取集合 有3個元素 為雙4連貫
```

```
            set_pop = set(lst) & set(cards)
            set_pop.remove('W')
            same_card = set_pop.pop()
            same_card2 = set_pop.pop()
            card, get_num, card2, get_num2 =  same_card, 4, same_card2, 4
        else: # 其他可能(3連貫+4連貫)
            if lst[0] == lst[1]:
                card, get_num, card2, get_num2 = lst[0], 4, lst[2], 3
            else:
                card, get_num, card2, get_num2 = lst[1], 4, lst[0], 3
    # 取得連貫資料和卡種 用卡牌賠率表card_score 計算得分!!
    score = card_score.loc[card][get_num-1] + card_score.loc[card2][get_num2-1]
    return score, lst #讀取卡牌賠率表輸出分數  另外同時輸出判斷式(作為消除的座標依據)

現在我們有算分數的函式 現在寫一個一次計算20條賠付線分數總和加上所有消除座標的函式, 方便可以計算消除,更新卡牌的功能!!
def spin_score(arr):#3x5卡牌 spin輸出一次 20條連貫 所有得分 及 消除卡片座標總計
    score = 0
    lst = []
    for n in range(1,21): # 20條線編號 迴圈計算20條條賠付線
        linscore, data = line_score(line(arr,n))
        if linscore > 0:
            score += linscore # 分數加總
            x_ind = line_way.iloc[n-1]  # 由連貫資料 換算賠付線中array位置(消除座標)
            for m in range(3):
                if data[m] != 'N':
                    lst.append((x_ind[m],m+1))
                    lst.append((x_ind[m+1],m+2))
                    lst.append((x_ind[m+2],m+3))
    lst = list(set(lst))  # 消除的座標取集合  過濾重複的資料~
    return score, lst

case1 & case2使用
得到消除的座標之後 此函式式消除座標內的位置 和要跟換數字
輸入資料
arr 原3x5 Array,
lst 需要刪除的座標
a 變更卡牌的數字(這裡設定為刪除為0, 而case2移動的W 則可以利用這個函式改為 1)
刪除的為0是方便更新計算  在下面會解釋
def clear_crd(arr, lst, a): # 消除卡片:n = 0   更換 W: n = 1
    for n, m in lst:
        arr[n-1][m-1] = a
    return arr
```

```
case1使用  卡片落下函式:
1.若最下排為0 下排和中排互換
2.若最中排為0 中排和上排互換
3.若最下排為0 下排和中排互換(第三步同第一步, 這是避免發生上中下是x, 0, 0的情況)
def drop_crd(arr): # 卡片落下
    for n in range(5):
        if arr[2,n] == 0:
            arr[2,n] = arr[1,n]
            arr[1,n] = 0
        if arr[1,n] == 0:
            arr[1,n] = arr[0,n]
            arr[0,n] = 0
        if arr[2,n] == 0:
            arr[2,n] = arr[1,n]
            arr[1,n] = 0
    return arr

case2使用, 尋找3x5 Array中W  並回傳座標位置
def find_W(arr): # 尋找3x5中的W 回傳卡片座標
    lst = []
    for n in range(3):
        for m in range(5):
            if arr[n][m] == 1: # W=1
                lst.append((n+1, m+1))
    return lst

case2使用, W的八方向隨機移動 移動代表W座標(x, y)
def move_W(): # 八方向移動 隨機選取 輸出移動座標
    mov = [(1,1),(1,0),(1,-1),(0,1),(0,-1),(-1,1),(-1,0),(-1,-1)]
    return mov[np.random.choice([0,1,2,3,4,5,6,7])]

case2使用, 輸入原W的座標(函式 find_W(arr) 的輸出)
輸出W移動後的座標  若是移動超出3x5 Array的範圍 刪除此座標
並輸出判斷值chick = 1 表示本次W移動出範圍, 方便後續使用
def after_mov(lst): # W 移動後座標
    lst_W=[]
    chick = 0
    for a, b in lst:
        mov = move_W() # 隨機移動座標
        lst_W.append((a+mov[0], b+mov[1])) # 新座標=原座標+隨機移動座標
    lst_W = list(set(lst_W)) # 檢查有相同的新座標整合為一
    if len(set(lst_W) - crdset) != 0: # 若有坐標系外的座標 len != 0
        chick = 1
    lst_W = list(set(lst_W) & crdset) # 整合在坐標系內的座標
    return lst_W, chick
```

case1 卡片更新函式, 這裡使用以上的函式,
輸入:
arr : 原3x5 Array
lst : 需要消除的座標
將3x5 Array 設定為數字的好處在此顯現, 方法是消除的座標變成0 ,而可以隨機生成一個新3x5 Array 並用bool判斷 保留0位置的隨機值
ex: array([[6, 2, 1, 2, 6],
[6, 8, 1, 10, 5],
[9, 4, 1, 7, 5]])
轉成卡牌樣式:
array([['E', 'A', 'W', 'A', 'E'],
['E', 'G', 'W', 'I', 'D'],
['H', 'C', 'W', 'F', 'D']], dtype='<U1')
由計算可知 消除座標為 : [(1, 2), (1, 3), (2, 3), (1, 4)]
消除後的3x5 Array
array([[6, 0, 0, 0, 6],
[6, 8, 0, 10, 5], (1)
[9, 4, 1, 7, 5]])
把以上Array 轉成bool值 是否等於0
array([[0, 1, 1, 1, 0],
[0, 0, 1, 0, 0], (2)
[0, 0, 0, 0, 0]])
生成一個新隨Array
array([[6, 7, 6, 5, 2],
[7, 3, 3, 10, 9], (3)
[1, 3, 9, 10, 10]])
將以上兩個Array相乘: (2) x (3)
array([[0, 7, 6, 5, 0],
[0, 0, 3, 0, 0], (4)
[0, 0, 0, 0, 0]])
將以上兩個Array相加: (1) x (4)
array([[6, 7, 6, 5, 6],
[6, 8, 3, 10, 5], (5)
[9, 4, 1, 7, 5]])
(5) 為更新的Array

```
def update_crd(arr, lst): # 更新卡片 消除掉落
    arr = clear_crd(arr, lst, 0)
    arr = drop_crd(arr)
    ar = spin()
    arr = ar*((arr == 0).astype(np.int)) + arr # arr 中 0的部分(消除的座標)換上新隨機數
    return arr
```

case2 卡片更新函式, 更新方法大致和case1相同, 但順序是找到W的座標, W座標移動一個位置,

```
def update_wmov(arr):# 更新卡片 W 移動
    lst_W = find_W(arr)
    af_W, chick =  after_mov(lst_W)
    arr = np.full((3, 5),0)
    arr = clear_crd(arr, af_W, 1)
    ar = spin()
    arr = ar*((arr == 0).astype(np.int)) + arr # arr 中 0的部分(消除的座標)換上新隨機數
    return arr, chick
```

3.

以上 是我們演算法要用的所有函式!!

case1 正式的演算法 檔案名稱 : SLOT1.py

```
import pickle   # 導入python 模組
import slot_func as slot
# 第一題 case1 消除掉落 演算法
allscore = 0
rtp_lst = []
spin_time = 500 # 500次記錄一次
for x in range(1,501):  # 總次數 500x500 25萬次
    for n in range(spin_time):
        arr = slot.spin()  # 一次spin 20條線都下注
        score , clr_lst = slot.spin_score(arr)  # 回傳分數,連貫的座標(消除座標)
        allscore += score  # 得分累積
        while score != 0:
            arr = slot.update_crd(arr, clr_lst)  # 消除掉落 更新牌組
            score , clr_lst = slot.spin_score(arr)  # 回傳分數.得分消除的座標
            allscore += score  # 得分累積
    print("chick %d"%x)
    rtp = allscore*100/(x*spin_time*20) # 20條線  100 百分比
    rtp_lst.append(rtp)
print('RTP:%.3f%%'%rtp)

pickle.dump(rtp_lst, open('case1_pickle.dat', 'wb')) # 資料儲存
```

結果有下圖呈現 :

4次25萬次spin 平均RTP 在約10萬次左右就達到了收斂:

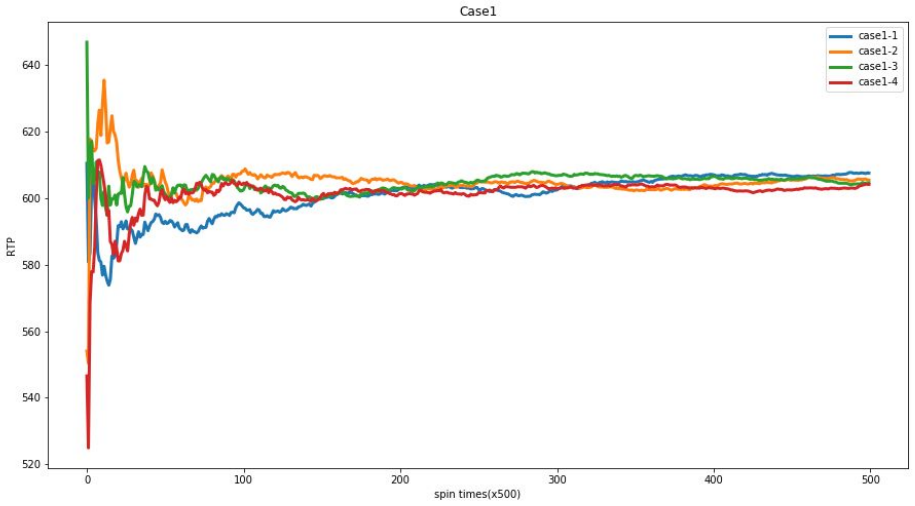
平均值為605%

RTP 資料分布

count　2000.00000　資料筆數(500次spin平均) 共100萬次

mean　 605.46612

std　 62.75112



4.

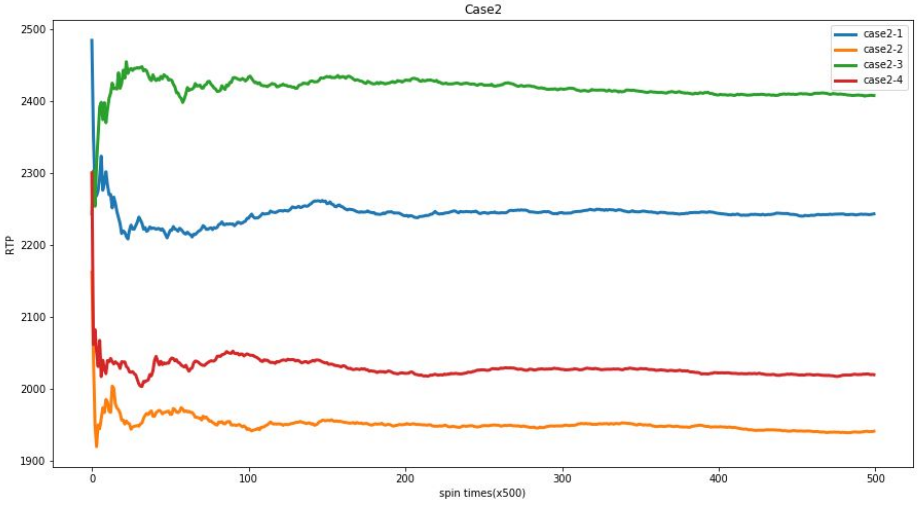
case2 正式的演算法 檔案名稱 : SLOT2.py

```
import pickle  # 導入python 模組
import slot_func as slot

# 第二題 case2 W 移動 演算法  沒有說明在沒有萬用卡的情況 計算得分是否會重新洗牌! 假設不會並結束回合

allscore = 0
rtp_lst = []
spin_time = 500 # 500次記錄一次
for x in range(1,251):  # 總次數 500x500 25萬次
    for n in range(spin_time):
        arr = slot.spin()  # 一次spin 20條線都下注
        chick = 0  # 判斷 W 有沒有跑出去框架
        score = slot.spin_score(arr)[0]  # 回傳分數
        allscore += score  # 得分累積
        if len(slot.find_W(arr)) != 0: # 沒有W 回合結束
            while chick == 0:  # 有W
                arr, chick = slot.update_wmov(arr) # 回傳更新牌組和W有沒有跑出框架判斷
                score = slot.spin_score(arr)[0]  # 回傳分數
                allscore += score  # 得分累積
    print("chick %d"%x)
    rtp = allscore*100/(x*spin_time*20) # 20條線  100 百分比
    rtp_lst.append(rtp)
print('RTP:%.3f%%'%rtp)
```

pickle.dump(rtp_lst, open('case2_pickle.dat', 'wb')) # 資料儲存

結果有下圖呈現 :
4次25萬次spin, 可已發現無法收斂, 表示資料離散隨機性很高,取平均分布來表示RTP
平均值為2152%　標準差 257%

RTP 資料分布
count　2000.000000　資料筆數(500次spin平均) 共100萬次
mean　 2152.876675
std　　257.805428



5. 結論:

Slot的機率在無限多種可能的情況~ 使用原型機模擬500次spin求的平均RTP值, 共取500次 25萬次!!
每個case 共做4次比對 共100萬次spin( ) 取得平均的RTP值
做4次可以比對是否有相同規律
原理是假設每次Spin都是一個離散機率密度分布, 由中央極限定理可以知道, 在離散的隨機分布多次疊加中, 機率密度會逐漸成形一個高斯分布的形況, 而高斯分布的最高值, 也是這個隨機分布的平均值, 就會是這個case中的平均分布值

若把所有可能列出 10^15 , 其實以100萬次的模擬遠遠小於所有樣本值!!
但是算出所有可能不切實際, 故整理一個統計型的答案