投稿類別- 自然科技(延伸組)

黑白大逆轉-棋盤變色破解與程式製作

作者
花蓮縣立自強國中 八年一班 張秉洋
花蓮縣立自強國中 八年一班 陳泰元

指導老師:
呂柏辰 老師
陳禹翔 老師

壹、前言

一、研究動機

在上次科展的研究中，我們知道了在(p,n)變色中有解的棋盤與其每個按鈕同餘$(n+p)$的按下次數，並找出$(1,n)$變色於$(n+1)\times(n+1)$棋盤的上下界，但上下界相差不小，因此我們想要找出其最小步數並做一個程式，使其只需輸入p與n的數值即可知道其最小步數及按法；我們原先的研究中所做出的遊戲程式之變色方式僅限於$(1,n)$變色，所以我們也想要做一個可自行選擇任意變色及棋盤的程式，而其中前者所產生的結果即是後者遊戲的最小步數按法。

二、研究目的

(一) 找出(p,n)變色於$(n+p)\times(n+p)$棋盤的最小步數解
(二) 做出可遊玩任意變色於任意棋盤的遊戲程式
(三) 做出可顯示任意變色於不同棋盤中的最小步數及各按鈕按下次數之最小步數的程式

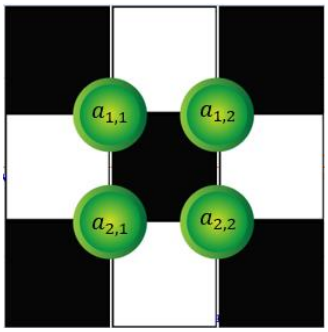
三、研究流程

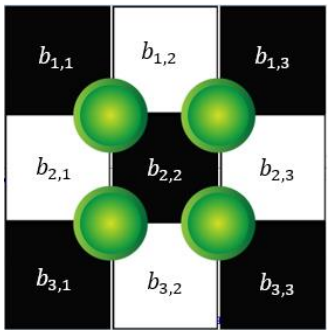
貳、正文

一、文獻探討

(一)名詞解釋：

1. 若方格位在棋盤中的第h列第k行，我們將其位置用$b_{h,k}$表示，根據按鈕位置的不同，我們將後續研究中，按鈕或其所按下次數記為$a_{n,m}$，如下圖。





2. 考慮原題中最終須將黑白互換，且遊戲規則為黑變白需按1次，白變黑需按2次，我們用序對「 $(1,2)$ 」表示，之後若遊戲規則變為黑變白需 i 次，白變黑需 j 次，則用序對(i,j)表示，定義為(i,j)變色。

3. 按鈕的同餘關係：
由於在變色的過程中，黑變白，白變灰，灰變黑，因此一個按鈕按三下後會將田字型回復原狀。換言之，若其中一個解須將按鈕按下t下，則將按鈕按下$t+3k(k\in\mathbb{N})$下仍為其解，即所有解除以三的餘數皆會相同，因此在後續的研究中將以同餘的方式討論。

(二)在科展的研究中，我們已經知道：

1. $(1,n)$變色僅在$(n+1)\times(n+1)$及其倍數棋盤有解，且其步數上下界分別為：

當 n 為奇數：

$$\begin{cases}\text{上界為}\dfrac{(n+1)^2(5n+1)}{8}\\ \text{下界為}\dfrac{(n+1)^2(3n-1)}{8}\end{cases}$$

當 $n\equiv 0(mod4)$：

$$\begin{cases}\text{上界為}\dfrac{(5n^2+11n+4)n}{8}\\ \text{下界為}\dfrac{(3n+5)n^2}{8}\end{cases}$$

當 $n\equiv 2(mod4)$：

$$\begin{cases}\text{上界為}\dfrac{(5n+6)(n^2+n+2)}{8}\\ \text{下界為}\dfrac{(3n+2)(n+2)(n-1)}{8}\end{cases}$$

2. (p,n)變色僅在$(n+p)\times(n+p)$及其倍數棋盤有解，且其按法如下所示：

					⋯	
$a_{1,1}$ $\equiv 1\times p$ $\equiv p$	$a_{1,2}$ $\equiv 1\times(-2p)$ $\equiv -2p$	$a_{1,3}\equiv 1\times 3p$ $\equiv 3p$	$a_{1,4}$ $\equiv 1$ $\times(-4p)$ $\equiv -4p$	$a_{1,5}$ $\equiv 1\times 5p$ $\equiv 5p$		$a_{1,n}$ $\equiv(-1)^{n+p-1}$ $\times np$
$a_{2,1}$ $\equiv -2\times p$ $\equiv -2p$	$a_{2,2}$ $\equiv(-2)(-2p)$ $\equiv 4p$	$a_{2,3}$ $\equiv(-2)\times 3p$ $\equiv -6p$	$a_{2,4}$ $\equiv -2$ $\times(-4p)$ $\equiv 8p$	$a_{2,5}$ $\equiv -2\times 5p$ $\equiv -10p$		$a_{2,n}$ $\equiv -2\times a_{1,n}$
$a_{3,1}$ $\equiv 3\times p$ $\equiv 3p$	$a_{3,2}$ $\equiv 3\times(-2p)$ $\equiv -6p$	$a_{3,3}\equiv 3\times 3p$ $\equiv 9p$	$a_{3,4}$ $\equiv 3\times(-4p)$ $\equiv -12p$	$a_{3,5}$ $\equiv 3\times 5p$ $\equiv 15p$		$a_{3,n}$ $\equiv 3\times a_{1,n}$
$a_{4,1}$ $\equiv -4\times p$ $\equiv -4p$	$a_{4,2}$ $\equiv(-4)(-2p)$ $\equiv 8p$	$a_{4,3}$ $\equiv(-4)\times 3p$ $\equiv -12p$	$a_{4,4}$ $\equiv -4$ $\times(-4p)$ $\equiv 16p$	$a_{4,5}$ $\equiv -4\times 5p$ $\equiv -20p$		$a_{4,n}$ $\equiv -4\times a_{1,n}$
$a_{5,1}$ $\equiv 5\times p$ $\equiv 5p$	$a_{5,2}$ $\equiv 5\times(-2p)$ $\equiv -10p$	$a_{5,3}\equiv 5\times 3p$ $\equiv 15p$	$a_{5,4}$ $\equiv 5\times(-4p)$ $\equiv -20p$	$a_{5,5}$ $\equiv 5\times 5p$ $\equiv 25p$		$a_{5,n}$ $\equiv 5\times a_{1,n}$
$a_{6,1}$ $\equiv -6\times p$ $\equiv -6p$	$a_{6,2}$ $\equiv(-6)(-2p)$ $\equiv 12p$	$a_{6,3}$ $\equiv(-6)\times 3p$ $\equiv -18p$	$a_{6,4}$ $\equiv -6$ $\times(-4p)$ $\equiv 24p$	$a_{6,5}$ $\equiv -6\times 5p$ $\equiv -30p$		$a_{6,n}$ $\equiv -6\times a_{1,n}$
⋯					⋯	⋯
$a_{n,1}$ $\equiv(-1)^{n+p-1}$ $\times pn$	$a_{n,2}$ $\equiv a_{n,1}(-2p)$	$a_{n,3}$ $\equiv a_{n,1}\times 3p$	$a_{n,4}$ $\equiv a_{n,1}$ $\times -4p$	$a_{n,5}$ $\equiv a_{n,1}\times 5p$	⋯	$a_{n,n}$ $\equiv a_{n,1}\times a_{1,n}$

二、研究過程

(一)程式設計

我們以原題及我們推廣的方向做出一個遊戲程式，並根據我們在先前的研究中發現的規律製作出它的破解版，且其顏色可自行制訂，訂製規則則是 RGB 輸入法，顏色和顏色之間使用斜線符號 (/)分開，首先是遊戲程式，其主要程式碼及遊戲畫面（以

1.匯入必要套件：

```
# 用來建立與操作棋盤矩陣
import numpy as np
# 用來繪圖顯示棋盤與按鈕
import matplotlib.pyplot as plt
# 建立互動式元件（輸入框、按鈕等）
import ipywidgets as widgets
# 顯示與清除輸出區
from IPython.display import display, clear_output
```

2.全域變數初始化：

```
board = None                          # 棋盤目前狀態
initial_board = None                  # 棋盤初始狀態（重設用）
valid_buttons = []                    # 可以點擊的按鈕座標
steps = 0                           # 步數計數器
```

```
cycle_rule = [(0.0,0.0,0.0),(1.0,1.0,1.0),(0.5,0.5,0.5)]
# 顏色循環規則 (黑→白→灰)
N = 3                                          # 棋盤大小 (N x N)
black_value = (0.0,0.0,0.0)                    # 黑格顏色 (RGB)
white_value = (1.0,1.0,1.0)                    # 白格顏色 (RGB)
button_color = (0.0, 1.0, 0.0)                 # 綠色按鈕 (RGB)
```

3.當只輸入不到三個數值時，將其補成三個數值：

```
def to_rgb_tuple(vals):
    if len(vals) == 1:
        # 如果只輸入一個數字 → 灰階
        return (vals[0], vals[0], vals[0])
    elif len(vals) == 2:
        # 如果輸入兩個數字 → (R,G,G)
        return (vals[0], vals[1], vals[1])
    elif len(vals) >= 3:
        # 如果輸入三個以上 → 取前三個 (R,G,B)
        return (vals[0], vals[1], vals[2])
    else:
        # 空輸入 → 黑色
        return (0.0,0.0,0.0)
```

4.翻譯使用者輸入之數值：

```
def set_cycle_rule(rule_str):
    default = [(0.0,0.0,0.0),(1.0,1.0,1.0)]  # 預設 黑→白
    parts = [s.strip() for s in rule_str.split("/") if
    s.strip() != ""]
    result = []
    for p in parts:
        try:
            vals = [float(x) for x in p.split(",") if
            x.strip()!=""]
            tup = to_rgb_tuple(vals)
            result.append(tup)
        except:
            pass
    return result if result else default
```

5.依循環規則回傳下個顏色

```
def cycle_value(val):
    if val not in cycle_rule:                  # 不在規則內 → 從頭開始
        return cycle_rule[0]
    idx = cycle_rule.index(val)                # 找到目前顏色的位置
    return cycle_rule[(idx + 1) % len(cycle_rule)]
                                               #循環到下一個顏色
```

6.尋找按下按鈕之相鄰方格：

```
def apply_button(r, c):
    global steps, board
```

```
    r0, c0 = r - 1, c - 1                    # 按鈕左上角對應的棋盤位置
    if r0 + 1 >= N or c0 + 1 >= N:           # 超出邊界 → 不處理
        return
    for dr in [0, 1]:                        # 影響四個格子
        for dc in [0, 1]:
            rr, cc = r0 + dr, c0 + dc
            board[rr, cc] = cycle_value(board[rr, cc])
    steps += 1                               # 步數 +1
    redraw_board()
```

7.重設棋盤：

```
def reset_board(_=None):
    global board, steps
    board = initial_board.copy()             # 回到初始狀態
    steps = 0
    redraw_board()
```

8.生成棋盤：

```
def generate_initial_board():
    b = np.zeros((N, N), dtype=object)       # 建立 NxN 棋盤矩陣
    for r in range(N):
        for c in range(N):
            if (r + c) % 2 == 0:             # 偶數格 → 黑
                b[r, c] = black_value
            else:                            # 奇數格 → 白
                b[r, c] = white_value
    return b
```

9.畫出棋盤和按鈕

```
# 建立程式的基本定義
def draw_board():
    fig, ax = plt.subplots(figsize=(5, 5))
    ax.set_xlim(0, N)
    ax.set_ylim(0, N)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect('equal')
    ax.invert_yaxis()
# 畫棋盤格子
    for r in range(N):
        for c in range(N):
            ax.add_patch(plt.Rectangle((c, r), 1, 1,
        color=board[r, c], ec='black', linewidth=2))
# 畫綠色按鈕 A[r,c]
    for r, c in valid_buttons:
        cx, cy = c, r
# 畫圓形按鈕
        circle = plt.Circle((cx, cy), 0.3, color=button_color)
        ax.add_patch(circle)
```

```
            ax.text(cx, cy, f'A{r},{c}', ha='center', va='center',
            fontsize=8, color='red')
    # 顯示步數
        ax.text(N/2, -0.3, f"{steps}", ha='center', va='center',
        fontsize=14, color='black')
        plt.show()
```

10.更新棋盤：

```
    def redraw_board():
        with board_output:
        clear_output(wait=True)                # 清除舊畫面
        draw_board()
```

11.變更參數：

```
    def update_board(_=None):
        global N, cycle_rule, white_value, steps, board,
        initial_board, valid_buttons, button_grid
        # 更新棋盤大小
        try:
            N = int(N_text.value)
        except:
            N = 3
        # 更新循環規則
        cycle_rule = set_cycle_rule(rule_text.value)
        # 更新白格顏色
        try:
            vals = [float(x) for x in white_text.value.split(",") if
            x.strip()!=""]
            white_value = to_rgb_tuple(vals)
        except:
            white_value = (1.0,1.0,1.0)
        # 重新建立棋盤
        steps = 0
        board = generate_initial_board()
        initial_board = board.copy()
        valid_buttons = [(r, c) for r in range(1, N) for c in
        range(1, N)]                           # 按鈕位置
        # 建立互動按鈕
        button_widgets = []
        for (r, c) in valid_buttons:
            def make_handler(r=r, c=c):
                def on_click(b):
                    apply_button(r, c)
                return on_click
            b = widgets.Button(description=f'A{r},{c}',
            layout=widgets.Layout(width='70px', height='30px'))
            b.on_click(make_handler(r, c))
            button_widgets.append(b)
        # 更新互動按鈕區
        button_grid.children = button_widgets
```

```
        button_grid.layout = widgets.Layout(grid_template_columns=
        f"repeat({N-1}, 80px)")
        redraw_board()

    N_text = widgets.Text(value="3", description="棋盤大小:")
    # 輸入棋盤大小
    rule_text = widgets.Text(value="0/1/0.5", description="循環規
    則:")
    # 輸入顏色循環規則
    white_text = widgets.Text(value="1", description="白格顏色:")
    # 輸入白格顏色
```

12.功能按鈕：

```
    update_button = widgets.Button(description="更新設定",
    layout=widgets.Layout(width='150px', height='30px'))
    reset_button = widgets.Button(description="重設棋盤",
    layout=widgets.Layout(width='150px', height='30px'))
```

13.棋盤顯示區塊：

```
    board_output = widgets.Output()
```

14.互動按鈕區塊：

```
    button_grid = widgets.GridBox()
    update_button.on_click(update_board)
    # 更新設定按鈕 → 更新棋盤
    reset_button.on_click(reset_board)
    # 重設棋盤按鈕 → 回到初始狀態
    control_box = widgets.VBox([N_text, rule_text, white_text,
    update_button, reset_button])
    display(control_box)                                    # 顯示輸入區
    display(button_grid)                                    # 顯示互動按鈕區
    display(board_output)                                   # 顯示棋盤區
```

15.畫初始棋盤：

```
    update_board()
```

(1,2)變色在3 × 3棋盤中的情況為例）如下：

破解版的程式及輸出畫面：

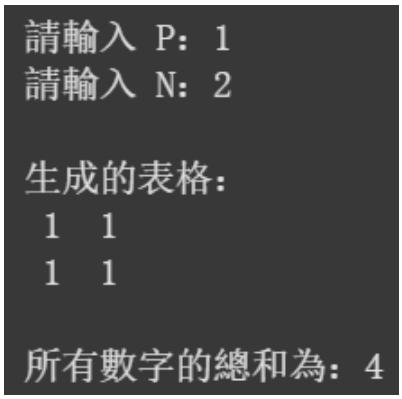
1.依照輸入大小回傳對應組合：

```
def compute_table(P, N):
size = P + N - 1            # 計算表格的邊長大小，等於 P + N - 1
table = []                  # 用來存放整張表格的二維清單
total_sum = 0               # 用來累積所有數字的總和
for r in range(size):       # 外層迴圈：逐列建立表格
row = []                    # 暫存當前這一列的數字
for c in range(size):
# 內層迴圈：逐行建立這一列的每個數字
      # 計算公式：根據列號 r、行號 c 以及 P 產生一個數字
      val = (((-1) ** (r + P)) * (r + 1)) * ((((-1)**
      (c + P)) * (c + 1)) * P)
      # 如果計算出來的數字 <= 0，就不斷加上 (size+1)，直到 > 0
      while val <= 0:
      val += size + 1
      # 如果數字大於 size，就不斷減去 (size+1)，直到 ≤ size
      while val > size:
            val -= size + 1
            row.append(val)     # 把修正後的數字加入當前列
            total_sum += val    # 同時累加到總和裡面
            table.append(row)   # 把這一列加到表格中
      return table, total_sum   # 回傳生成好的表格和總和
if __name__ == "__main__":      # 只有當這支程式直接執行時才會執行
```

2.詢問並呼叫函式程式碼：

```
P = int(input("請輸入 P："))          # 從使用者輸入整數 P
N = int(input("請輸入 N："))          # 從使用者輸入整數 N
table, total = compute_table(P, N)   # 呼叫函式生成表格和總和
width = len(str(max(max(row) for row in table))) + 1
# 設定每個數字輸出的寬度（對齊用）
print("\n 生成的表格：")                # 輸出標題
      for row in table:               # 逐列印出表格
print(" ".join(f"{val:{width}}" for val in row))
# 每個數字用固定寬度排版
print(f"\n 所有數字的總和為：{total}")   # 最後輸出總和
```

以(1,2)變色中的3 × 3棋盤及(3,8)變色中的11 × 11棋盤為例：



參、結論

一、 $(1,n)$變色僅在$(n+1)\times(n+1)$棋盤中有解，且其步數上下界如下：
當 n 為奇數：

$$\begin{cases}\text{上界為}\dfrac{(n+1)^2(5n+1)}{8}\\ \text{下界為}\dfrac{(n+1)^2(3n-1)}{8}\end{cases}$$

當 $n\equiv 0(mod4)$：

$$\begin{cases}\text{上界為}\dfrac{(5n^2+11n+4)n}{8}\\ \text{下界為}\dfrac{(3n+5)n^2}{8}\end{cases}$$

當 $n\equiv 2(mod4)$：

$$\begin{cases}\text{上界為}\dfrac{(5n+6)(n^2+n+2)}{8}\\ \text{下界為}\dfrac{(3n+2)(n+2)(n-1)}{8}\end{cases}$$

二、 在(p,n)變色中，僅$(p+n)\times(p+n)$及其倍數邊長之棋盤有解，且其按鈕按法如下：

1. 當$t<n+p$時，$t\times t$棋盤無解。
2. 僅$(n+p)k\times(n+p)k$棋盤有解。
3. (p,n)變色在$(n+p)\times(n+p)$棋盤中的解為

$$\begin{cases}\text{第一列：}a_{1,n}\equiv(-1)^{n+1}\times pn(mod\ n+p)\\ \text{第一行：}a_{n,1}\equiv(-1)^{n+1}\times pn(mod\ n+p)\\ \text{其餘按鈕：}a_{m,l}\equiv a_{m,1}\times a_{1,l}\end{cases}$$

三、 將原題製作成遊戲程式是可行的，且其破解版也可以被製作出來。
四、 在遊戲中，玩家可自行設定顏色及變色規則。
五、 只要將變色規則數入破解版程式中，只要依破解版中所輸出的表格，一一對應到遊戲版中，便能使其成功變色。

肆、引註資料

一、 黑白大逆轉：棋盤變色任務：
https://reurl.cc/6qR3Ld
二、 Python 教學基礎篇：初學者「一定要會」的基本語法與符號：
https://reurl.cc/lYlARQ
三、 國中資優數學之同餘的探討：
https://hdl.handle.net/11296/xu7drf
四、 厲害的人會這樣下指令！Google 資深 AI 工程師公開 10 種用法：
https://reurl.cc/oYXLWQ
五、 新時代學 Python、AI 的最佳方法：

https://reurl.cc/vLoA7k