新北市立板橋高級中學

學生自主學習計畫

研究成果報告書

Python 程式設計實作—俄羅斯方塊

研 究 生：三年十三班 王承琳

中 華 民 國 一 百 一 十 一 年 三 月

摘要

本研究使用程式設計製作俄羅斯方塊遊戲，透過研究 Python 不同函式的指令跟運算來訓練邏輯思考及對於 Python 語言的運用能力，並學習如何找出問題、分析問題進而解決問題、完成專案。

研究心得

研究初期時，我對於研究步驟沒有很清楚的規劃，導致我像隻無頭蒼蠅般，寫出來的程式雜亂無章，邏輯錯亂，所以我花了一部份時間進行步驟規劃以及程式的重整，將我所想要的遊戲效果或功能列點並分為不同系統，在根據此去依序進行。當遊戲的基礎功能都完成後，我會請同學試玩來測試遊戲是否有錯誤並進行修改，而在 Debug 時，我發現備註的重要性，若沒有在打程式時善用備註紀錄思考方向，在後續修改時會花很多時間來理解當初的想法以及思路，因此我在之後的程式碼都有打備註，並將之前的備註補上，且我也發現寫程式真的需要非常多的耐心以及細心，因為每個系統都是環環相扣的，一步錯，步步錯，所以在修改時須耐心地不斷檢查各個部分錯誤的地方，有時候粗心少打一個空格都會造成程式的錯誤。在後續重新整理程式碼時，我注意到我許多太過於繁複且沒必要的步驟，雖然最後有達到想要的效果，但卻是轉了一個彎才達成，也就是說我邏輯不夠直線，常造成程式太過冗長，就像寫作文中的贅字一樣，邏輯夠不夠清楚可以間接影響專案的完成速度以及精簡度，所以我將程式碼全部重整過，盡量修改成最精簡的樣子，刪去不必要的變數或迴圈等，並重新順一次思路，建立邏輯思考能力的基礎。

目錄

摘要 1
研究心得 1
目錄 2
一、研究背景與動機 3
1.1 研究背景與動機 3
1.2 研究期許 3
二、緒論 4
2.1 俄羅斯方塊起源 4
2.2 俄羅斯方塊基本圖形與規則 4
三、研究方法 5
3.1 研究流程 5
3.2 研究工具 5
四、程式設計實作 6
4.1 初始化 6
4.2 座標系統 8
4.3 移動系統 11
4.4 旋轉系統 25
4.5 消除系統 26
4.6 畫面顯示 30
4.7 音效 34
4.8 總體設計概念 35
五、結論與建議 36
5.1 結論 36
5.2 建議 37
資料來源 38

第一章 研究背景與動機

1.1 研究背景與動機

在我國中小時，Facebook 裡有一款遊戲叫做 Tetris Battle，當時風靡全校，幾乎只要有使用 Facebook 的同學都有在玩這款遊戲，每天放學後的娛樂之一就是與好友進行對戰。而在我國三時，因為即將到會考的日子了，所以我沒有再玩過這款遊戲，直到升上高中後突然憶起這款經典遊戲時，它已經結束營運了。

我想要嘗試著將我腦海中的那個經典重現，故我選擇了此主題，使用程式語言 Python 製作出一款俄羅斯方塊遊戲，除了可以回味小時候的感動，更可以精進程式設計的能力。

1.2 研究期許

我希望我能夠透過此研究訓練邏輯思考能力及如何針對問題做出改善，利用 Python 的功能來達到我所想要的效果。此外，就獨立思考而言，我希望我能夠透過設計這款遊戲訓練解決問題的能力，就學術研究而言，我希望我的研究能夠幫助到其他對程式設計同樣有興趣的學生們，就專業領域而言，我希望我能夠精進程式設計的運用能力，將課堂所學化為己用，比起課堂上練習的題目，自己獨立完成一款遊戲的難度與挑戰性都高了許多，能進步的幅度也大了許多。

第二章 緒論

2.1 俄羅斯方塊起源

此遊戲是由一位名為阿列克謝·帕基特諾夫的蘇聯人所設計，約於 1984 年首次發佈。此遊戲的名稱「Tetris」是由希臘語「四」的前綴「tetra-」（因所有落下方塊皆由四塊組成）和「tennis」（作者最喜歡的運動）拼接而成，而華語地區則因為作者是俄羅斯人所以將其稱為「俄羅斯方塊」。

2.2 俄羅斯方塊基本圖形與規則

一、基本圖形：最基本的俄羅斯方塊圖形由四個正方形所組成，共有七種組合。

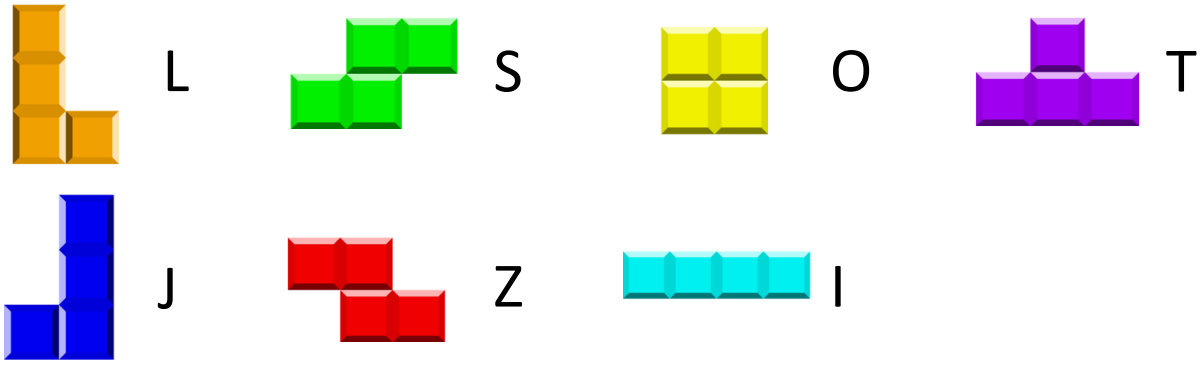

圖 2.1、俄羅斯方塊的七種圖形

(資料來源：維基百科－俄羅斯方塊)

二、規則

(一) 玩家操作

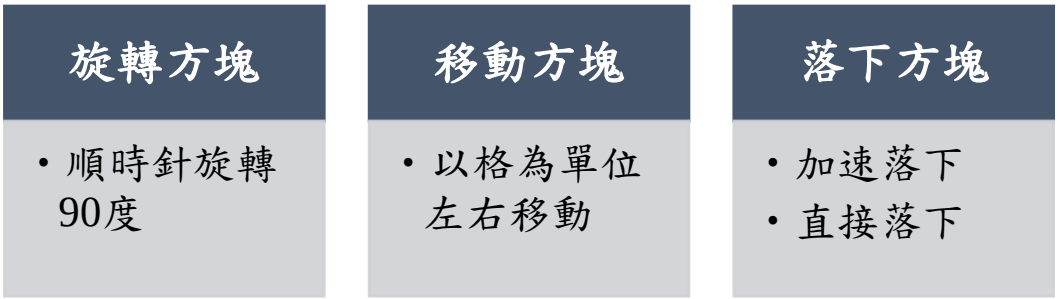

圖 2.2、玩家操作的三種類型

(二) 得分方式

當區域中某一橫列格子全部被方塊填滿時，則玩家得分並**消除該層方塊**。

當方塊堆疊到遊戲區域頂端而無法繼續消除層數時，則**判定遊戲結束**。

分數若越高下落速度就越快，且每消除一層方塊的得分也越多。

第三章 研究方法

3.1 研究流程

研究者先訂定所需完成的功能，並分為 5 個部分來進行。

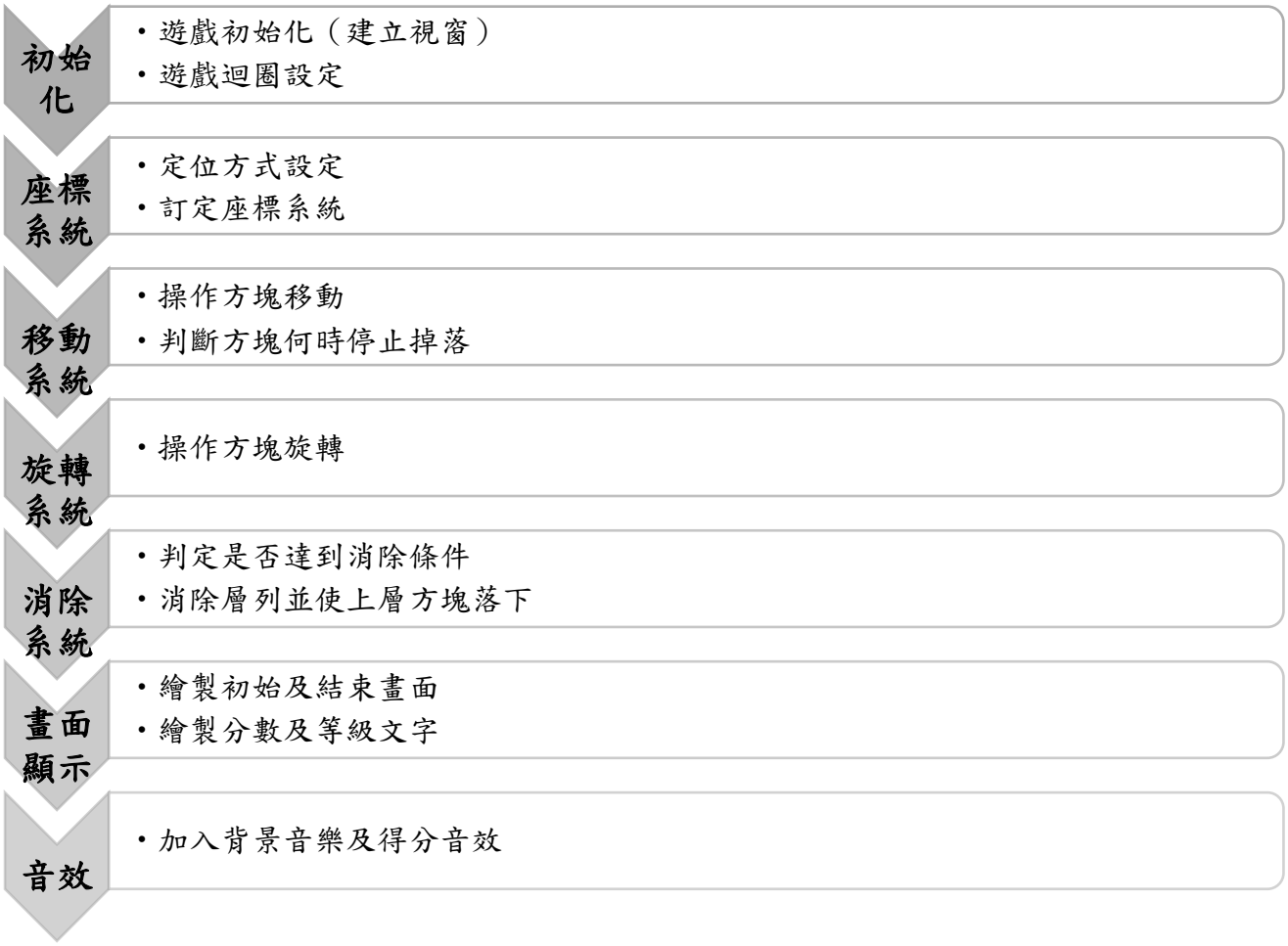

圖 3.1、研究流程圖

(資料來源：研究者自行繪製)

3.2 研究工具

研究工具為 Visual Studio Code，使用的程式語言為 Python 3.9.6 及 Python 內鍵模組 Pygame 2.1.2，使用 Pygame 模組前需先引入（圖 3.2），圖中 random 及 os 為後續所需模組。

```
1   import pygame
2   import random
3   import os
```

圖 3.2、引入模組

第四章 程式設計實作

4.1 初始化

一、 Pygame 初始化及視窗建立

起初先進行 Pygame 初始化才能進行後續動作。

```
25  WIDTH = 900
26  HEIGHT = 700
27
28  #遊戲初始化
29  pygame.init()
30  #創建視窗
31  screen = pygame.display.set_mode((WIDTH, HEIGHT))
32  #更改檔名
33  pygame.display.set_caption("TETRIS")
34  #創建物件(管理操控時間)
35  clock = pygame.time.Clock()
```

圖 4.1、遊戲初始化及基本設定

二、 遊戲迴圈

遊戲要一直持續進行，所以建立一個 while 迴圈，作為遊戲持續運行迴圈（遊戲主迴圈）。

```
242 BAR_WIDTH = 300
243 BAR_HEIGHT = 600
244
245 BLACK = (0, 0, 0)
246 LIGHT_GRAY = (128, 128, 128)
247
248 running = True
249 FPS = 125
250
251 while running:
252     #1秒鐘內最多執行幾次
253     clock.tick(FPS)
254
255     #取得輸入
256     for event in pygame.event.get():
257         if event.type == pygame.QUIT:
258             running = False
259
260     #畫面顯示
261     screen.fill(BLACK)
262     outline_rect = pygame.Rect(100, 50, BAR_WIDTH, BAR_HEIGHT)
263     pygame.draw.rect(screen, LIGHT_GRAY, outline_rect, 2)
264     for i in range(9):
265         pygame.draw.line(screen, LIGHT_GRAY, (130+i*30, 50), (130+i*30, 648), 2)
266     for i in range(19):
267         pygame.draw.line(screen, LIGHT_GRAY, (100, 80+i*30), (398, 80+i*30), 2)
268
269     pygame.display.update()
270
271 pygame.quit()
```

圖 4.2、遊戲主迴圈

程式說明

[256-258]若點擊關閉視窗，則退出迴圈。

[261-269]將視窗填滿黑色，繪製遊戲區域並更新畫面顯示（圖 4.3）。

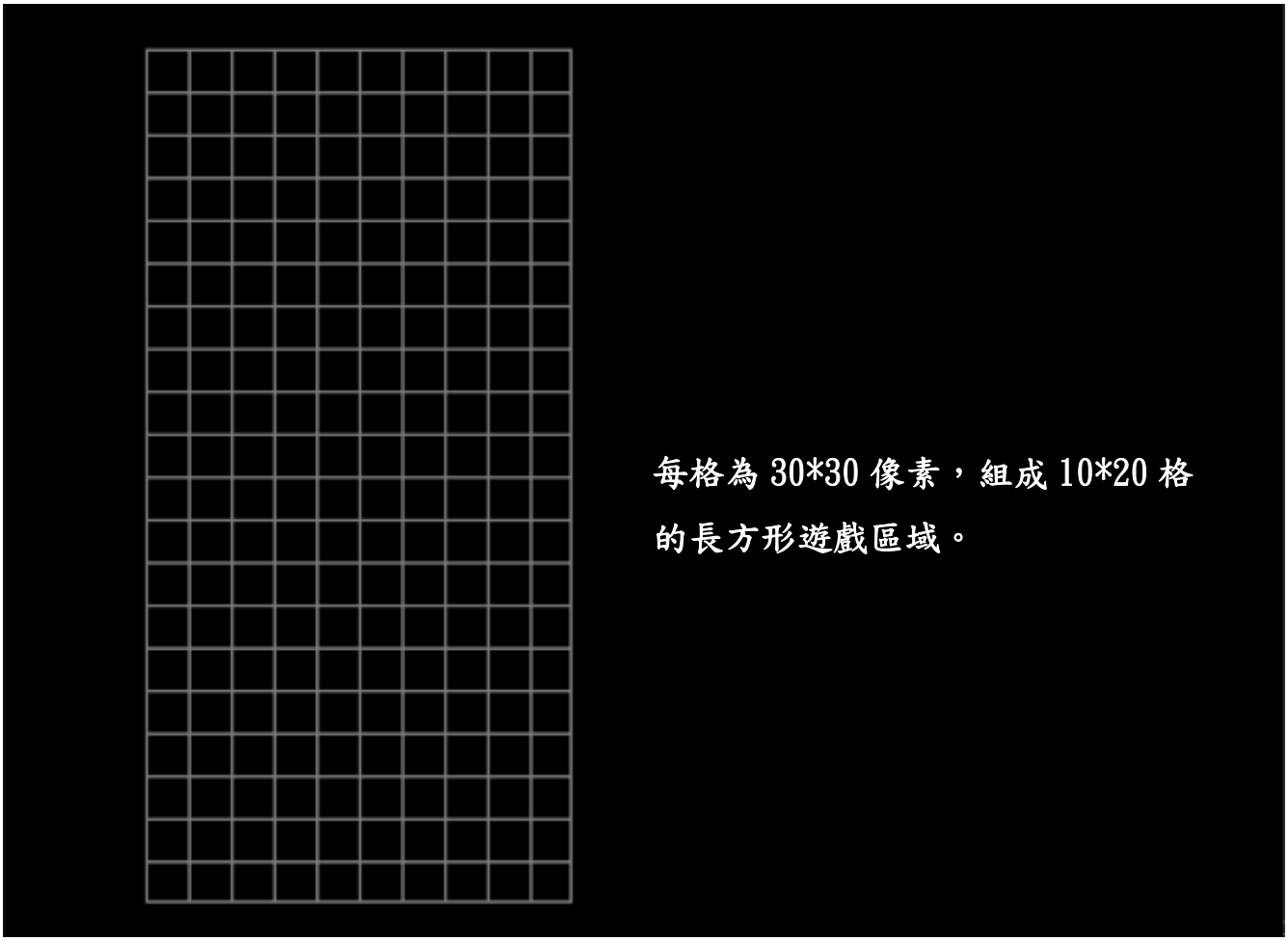

圖 4.3、遊戲畫面繪製範例圖

設計說明

現今大部分俄羅斯方塊的遊戲區域皆設定為長方形，在繪製遊戲畫面時，我有考慮過使用其他圖形，例如正方形抑或是星型等特殊圖形，但顧及到以下幾點，最後我還是選了長方形來做為遊戲區域。

(一) 須預留足夠的高度使方塊不要太快落地

(二) 若左右太寬，玩家需要花較久的時間才能達到消除條件

(三) 若使用特殊圖形作為遊戲區域，將無法用方塊完全填滿

4.2 訂定座標系統

一、定位方式

為了達成七種不同圖形皆以同種方式定位，透過觀察各圖形的共通點為起點，俄羅斯方塊各圖形皆由四個小正方形組成，由於當中旋轉後掃過佔最大面積的為 I 型方塊（圖4.4），故以此為標準，分為16個格子，標為0-15。

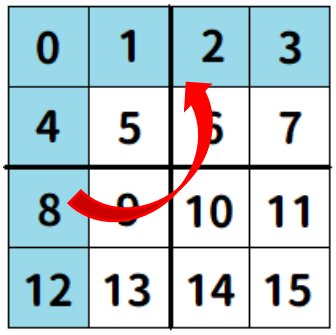

圖 4.4、I 型方塊編號圖

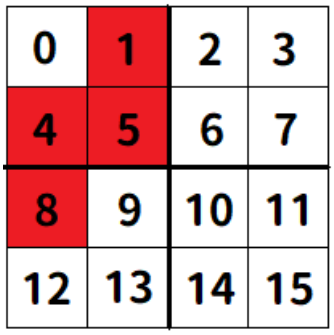

圖 4.5、Z 型方塊編號圖

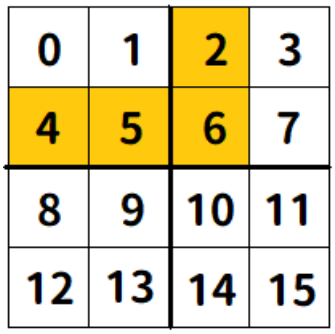

圖 4.6、L 型方塊編號圖

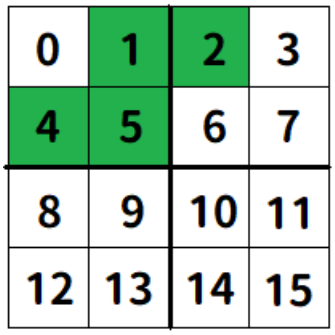

圖 4.7、S 型方塊編號圖

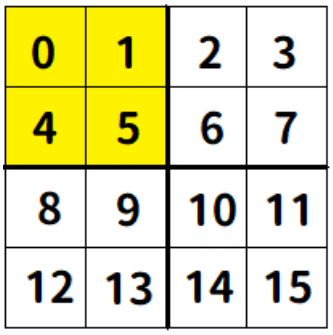

圖 4.8、O 型方塊編號圖

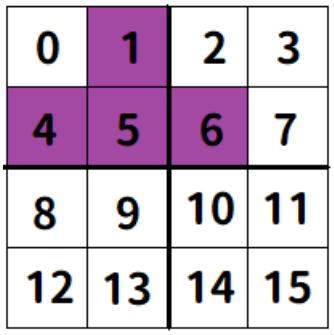

圖 4.9、T 型方塊編號圖

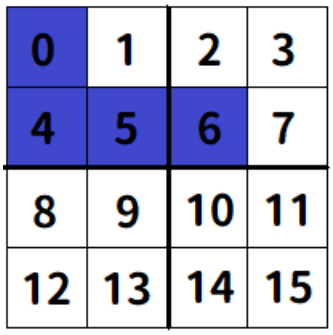

圖 4.10、J 型方塊編號圖

(資料來源：研究者自行繪製)

二、訂定座標

依此標準我們可以將這 7 種圖形及其旋轉後的圖形定義為 4 個編號。

```
35  #方塊建立(訂定座標)
36  T = [[1, 4, 5, 6],
37       [0, 4, 5, 8],
38       [0, 1, 2, 5],
39       [1, 4, 5, 9]]
40  S = [[1, 2, 4, 5],
41       [0, 4, 5, 9]]
42  Z = [[1, 4, 5, 8],
43       [0, 1, 5, 6]]
44  I = [[0, 4, 8, 12],
45       [4, 5, 6, 7]]
46  J = [[0, 4, 5, 6],
47       [0, 1, 4, 8],
48       [0, 1, 2, 6],
49       [1, 5, 8, 9]]
50  L = [[2, 4, 5, 6],
51       [0, 4, 8, 9],
52       [0, 1, 2, 4],
53       [0, 1, 5, 9]]
54  O = [[0, 1, 4, 5]]
55
56  all_objects = [T, Z, J, I, L, O, S]
57  all_colors = [PURPLE, RED, DEEP_BLUE, LIGHT_BLUE, ORANGE, YELLOW, GREEN]
```

```
16  #color name = (R, G, B)
17  ORANGE = (255, 165, 0)
18  PURPLE = (153, 50, 204)
19  GREEN = (0, 255, 0)
20  RED = (255, 0, 0)
21  YELLOW = (255, 255, 0)
22  LIGHT_BLUE = (0, 245, 255)
23  DEEP_BLUE = (72, 61, 139)
```

圖 4.11、方塊編號序列

程式說明

[56]建立新序列命名為 all_objects 並將七種方塊的序列存入。

→方便後續隨機選擇初始方塊及旋轉方塊（各方塊序列按照順時針旋轉的順序排列）。

[57] 建立新序列命名為 all_colors 並將七種方塊的對應顏色存入。

定義完方塊編號後，再將各編號的初始座標存入字典中。

```
63  class Position():
64      def __init__(self):
65          #各編號方塊出現的初始座標[x, y]
66          self.all_spaces = {0:(220, -10), 1:(250, -10), 2:(280, -10), 3:(310, -10),
67                             4:(220, 20), 5:(250, 20), 6:(280, 20), 7:(310, 20),
68                             8:(220, 50), 9:(250, 50), 10:(280, 50), 11:(310, 50),
69                             12:(220, 80), 13:(250, 80), 14:(280, 80), 15:(310, 80)}
70          #各編號方塊在4*4格子中的[列, 行]
71          self.mn_dic = {0:[0, 0], 1:[0, 1], 2:[0, 2], 3:[0, 3],
72                         4:[1, 0], 5:[1, 1], 6:[1, 2], 7:[1, 3],
73                         8:[2, 0], 9:[2, 1], 10:[2, 2], 11:[2, 3],
74                         12:[3, 0], 13:[3, 1], 14:[3, 2], 15:[3, 3]}
```

```
228  position = Position()
```

圖 4.12、方塊編號的初始座標字典

程式說明

[63]建立類別 Position 存放座標系統的屬性及函式。

[64-74]建立初始函數並存入各編號方塊出現的初始座標（all_spaces）及其在 4*4 格子中的[列, 行]（mn_dic）（圖 4.13）→方便後續繪製圖形

[228]將變數 position 加入 Position 類別

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

圖 4.14、[列, 行]標示圖

4.3 移動系統

一、 直線落下

建立一個 Move 類別存放所有移動系統的屬性及函式，可讓程式碼更清楚明瞭。

```
76  class Move():
77      def init(self):
78          self.speed = 0.5
```

```
230 move = Move()
231 move.init()
```

圖 4.15、重置遊戲初始值函式

程式說明

[77-78]建立 init 函式存放遊戲初始值（後續會再放入其他數值）。

[230-231]將變數 move 加入 Move 類別並執行 init 函式重置遊戲初始值。

每個初始方塊初始值（座標、顏色等）不同，故建立函式來進行數值重設。

```
76  class Move():
126     def draw_init(self):
127         #隨機選擇初始圖形
128         self.choose = random.choice(all_objects[0:7])
129         self.n = random.choice(self.choose)
130         self.color = all_colors[all_objects.index(self.choose)]
131         #定位原點
132         self.O = [-2, 4]
133         #方塊座標y值增加數值
134         self.speed_n = 0
135         self.SPEEDy = 0
```

圖 4.16、重置方塊初始值函式

程式說明

[126-135]建立 draw_init 函數存放每個初始方塊的初始值

[128]隨機選擇圖形種類　[129]隨機選擇圖形旋轉態。

[130]獲取圖形對應的顏色　[132]定位原點座標[列, 行](遊戲區域的格數)。

[134]方塊座標 y 值增加的次數　[135]方塊座標 y 值增加的總值。

建立完重製方塊初始值的函式後，跟重置遊戲初始值的函式一起執行。

```
76  class Move():
77      def init(self):
78          self.speed = 0.5
79          self.draw_init()
```

圖 4.17、重置遊戲、方塊初始值

程式說明

[82]將 draw_init 函式加入 init 函式中，則當遊戲重新開始後會重置初始方塊的數值。

建立完初始值後，將方塊繪製在畫面上。

```
76  class Move():
```

```
103     def draw(self):
104         if (self.speed*self.speed_n) % 30 == 0:
105             self.SPEEDy = (self.speed)*self.speed_n
106             self.O[0] += 1
107         for j in range(len(self.n)):
108             self.X, self.Y = position.all_spaces[self.n[j]]
109             self.Y += self.SPEEDy
110             if self.Y >= 50:
111                 fill_rect = pygame.Rect(self.X, self.Y, 30, 30)
112                 pygame.draw.rect(screen, self.color, fill_rect)
113         self.speed_n += 1
```

圖 4.18、繪製方塊函式

程式說明

[103-113]建立 draw 函式繪製方塊。

[104-106]每運行一次此函式，方塊 y 值增加次數加 1，當次數乘初始速度為 30 的倍數時，則方塊 y 值增加總值加上其值且定位原點的列數加 1。

[107-112]透過迴圈進行方塊中四個小正方形（之後將其稱為「小方塊」）的繪製（[110]若超出遊戲邊界上方則不進行繪製）。

設計說明

因小方塊初始座標在迴圈內皆須重置，則 y 座標會回到初始位置，為了實現讓每個方塊一起下落（即 y 座標持續增加），故我將方塊 y 座標增加總值獨立為一個迴圈外的變數，則只要在方塊 y 座標重置後再加上增加總值，即可實現目的。且因此函式會在遊戲主迴圈內進行，而主迴圈有限制的運行效率（FPS），為了實現方塊下落時以一格為單位，所以方塊 y 座標應一次增加 30 像素，但若直接設定每執行一次 draw 函式就將 y 座標增加 30 像素，下落速度會太快，因此我建立 speed_n 變數存放方塊座標 y 值增加的次數，就可以使方塊以一格為單位下落且不會造成速度過快。

建立完繪製圖形函式後，在遊戲主迴圈內進行方塊繪製。

```
245  while running:
```

```
271      #畫面顯示
272      screen.fill(BLACK)
273      move.draw()
274      for i, j in move.imgs:
275          screen.blit(i, j)
276      outline_rect = pygame.Rect(100, 50, BAR_WIDTH, BAR_HEIGHT)
277      pygame.draw.rect(screen, LIGHT_GRAY, outline_rect, 2)
278      for i in range(9):
279          pygame.draw.line(screen, LIGHT_GRAY, (130+i*30, 50), (130+i*30, 648), 2)
280      for i in range(19):
281          pygame.draw.line(screen, LIGHT_GRAY, (100, 80+i*30), (398, 80+i*30), 2)
282
283      pygame.display.update()
284
285  pygame.quit()
```

圖 4.19、進行方塊繪製

程式說明

[273]在遊戲主迴圈內執行 draw 函式。

二、 左右移動

左右移動改變的是 x 座標，因此要建立方塊 x 座標增加數值的變數。

```
76   class Move():
```

```
131      def draw_init(self):
132          #隨機選擇初始圖形
133          self.choose = random.choice(all_objects[0:7])
134          self.n = random.choice(self.choose)
135          self.color = all_colors[all_objects.index(self.choose)]
136          #定位原點
137          self.O = [-2, 4]
138          #方塊座標y值增加數值
139          self.speed_n = 0
140          self.SPEEDy = 0
141          #方塊座標x值增加數值
142          self.SPEEDx = 0
```

圖 4.20、方塊 x 座標增加數值的變數

程式說明

[142]在 draw_init 函式中建立 SPEEDx 變數存放方塊座標 x 值增加數值的初始值。

建立一函式進行左右移動（改變方塊 x 座標）。

```
76   class Move():
```

```
160      def move(self, direction):
161          if direction == "R":
162              self.O[1] += 1
163              self.SPEEDx += 30
164          if direction == "L":
165              self.O[1] -= 1
166              self.SPEEDx -= 30
```

圖 4.21、左右移動函式

程式說明

[160-166]建立 move 函式，輸入移動方向（左、右）。

[161-166]若方向輸入”R”，則圖形右移一格；反的，若方向輸入”L”，則圖形左移一格。

若左右移動或是後續的旋轉方塊沒有限制邊界，可能會造成方塊超出遊戲區域的情況，故建立一函式來檢查方塊是否超出遊戲區域並將其修正到正確位置。

```
76    class Move():
```

```
130       def region_judge(self):
131           self.judge_R = []
132           for i in self.n:
133               self.judge_R.append(self.O[1]+position.mn_dic[i][1])
134           #超出右邊界線
135           if max(self.judge_R) > 9:
136               self.O[1] -= max(self.judge_R)-9
137               self.SPEEDx -= 30*(max(self.judge_R)-9)
138           #超出左邊界線
139           if min(self.judge_R) < 0:
140               self.O[1] -= min(self.judge_R)
141               self.SPEEDx -= 30*min(self.judge_R)
```

圖 4.22、檢查方塊是否超出遊戲區域

程式說明

[130-141] 為了避免圖形左右移動或旋轉後超出遊戲邊界，故建立 region_judge 函式判定是否超過遊戲邊界。

[131-133]將圖形中方塊目前所在行數加入序列，最大值為圖形最右邊的行數，最小值為圖形最左邊的行數。

[135-137]若圖形最右邊的行數大於 9，即超出右邊界線，則圖形左移一格。

[139-141]若圖形最左邊的行數小於 0，即超出左邊界線，則圖形右移一格。

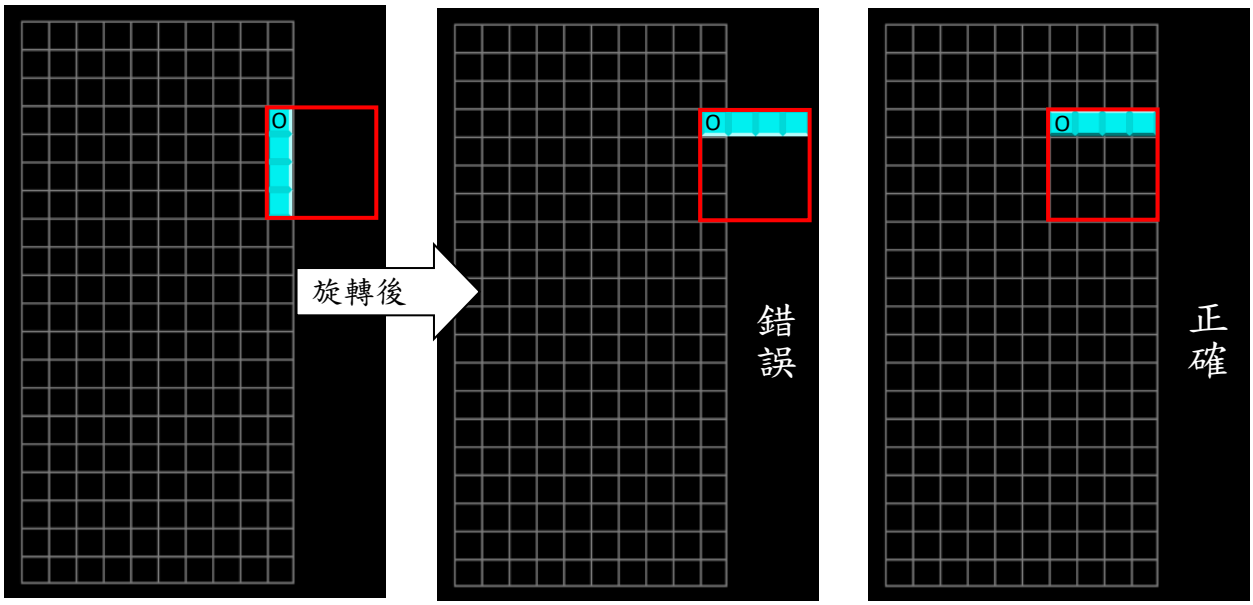

圖 4.23、方塊旋轉後正確及錯誤位置示意圖

(資料來源：研究者自行繪製)

按下左右鍵時，執行左右移動函式。

```
245  while running:
```

```
270      #取得輸入
271      for event in pygame.event.get():
272          if event.type == pygame.QUIT:
273              running = False
274          if event.type == pygame.KEYDOWN:
275              #移動系統
276              if event.key == pygame.K_RIGHT:
277                  move.move("R")
278              if event.key == pygame.K_LEFT:
279                  move.move("L")
```

圖 4.24、檢查是否按下左右鍵

程式說明

[274-279]在遊戲主迴圈內加入判斷式，若按下左、右鍵則執行 move 函式。

在繪製方塊之前先檢查方塊是否超出遊戲區域並修正，然後改變方塊 x 座標，則繪製出來的方塊就會在正確位置了。

```
76   class Move():
```

```
101      def draw(self):
102          if (self.speed*self.speed_n) % 30 == 0:
103              self.SPEEDy = (self.speed)*self.speed_n
104              self.O[0] += 1
105          self.region_judge()
106          for j in range(len(self.n)):
107              self.X, self.Y = position.all_spaces[self.n[j]]
108              self.X += self.SPEEDx
109              self.Y += self.SPEEDy
110              if self.Y >= 50:
111                  fill_rect = pygame.Rect(self.X, self.Y, 30, 30)
112                  pygame.draw.rect(screen, self.color, fill_rect)
113          self.speed_n += 1
```

圖 4.25、檢查方塊是否超出遊戲區域並改變方塊 x 座標

程式說明

[105]在 draw 函式中加入 region_judge 函式（加在迴圈外，即可移動整個圖形）。

[108] 方塊 x 座標加上需增加的總值。

三、 正確位置停下

為了判斷圖形應停下的位置，須將哪格有方塊填滿記錄下來。

```
76  class Move():
```

```
77      def init(self):
78          self.speed = 0.5
79          self.draw_init()
80          self.lines = []
81          for i in range((HEIGHT-100)//30):
82              self.lines.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
83          for i in range(3):
84              self.lines.append([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

圖 4.26、紀錄方塊是否被填滿

程式說明

[80-84]在 init 函式中建立一個二維序列存放每一格是否被填滿（0 為未填滿；1 為已填滿）。

設計說明

應我採取以 4*4(格)的範圍來定位，也是以 4*4(格)的範圍來檢查，故若我只建立 20 列（遊戲區域共 20 列），則當檢查到倒數第三列時，接下來的列數不足 4 列，會造成錯誤，故要再增加三列並設定為填滿狀況以便後續依列檢查。

紀錄完方塊填滿狀況後，建立 judge_touch 函式判斷方塊應在何時停下。

```
76  class Move():
```

```
137     def judge_touch(self):
138         self.judge_list = []
139         for i in range(0, 19):
140             judge_objects = 0
141             for h in self.n:
142                 if self.lines[i+position.mn_dic[h][0]][self.O[1]+position.mn_dic[h][1]] == 0:
143                     judge_objects += 1
144             if judge_objects == 4:
145                 self.judge_list.append(True)
146             else: self.judge_list.append(False)
```

圖 4.27、判斷方塊應在何時停下函式

程式說明

[138]建立 judge_list 序列存放布林值。

[139-146]透過迴圈檢查在每一列是否可以作為圖形停下的定位線。

使方塊在定位線停下。

```
76    class Move():
```

```
102       def draw(self):
103           if (self.speed*self.speed_n) % 30 == 0:
104               self.SPEEDy = (self.speed)*self.speed_n
105               self.O[0] += 1
106           self.region_judge()
107           for j in range(len(self.n)):
108               self.X, self.Y = position.all_spaces[self.n[j]]
109               self.X += self.SPEEDx
110               self.Y += self.SPEEDy
111               self.judge_touch()
112               if self.O[0] >= 0: O = self.O[0]
113               else: O = 0
114               for i in range(O, len(self.judge_list)):
115                   if self.judge_list[i] == False:
116                       self.stop_line = i-1
117                       break
118                   if self.judge_list.count(True) == 19:
119                       self.stop_line = 18
120               if self.Y >= 50:
121                   fill_rect = pygame.Rect(self.X, self.Y, 30, 30)
122                   pygame.draw.rect(screen, self.color, fill_rect)
123           self.speed_n += 1
```

圖 4.28、判斷定位線

程式說明

[111]將 judge_touch 函式加入 draw 函式。

[112-119]若從圖形定位原點所在的列數往下到某列檢查結果皆是 True，則此列就為定位線。

設計說明

初始圖形定位原點會在遊戲區域外（－2），故檢查時若定位原點列數小於 0，就須從 0 開始檢查（否則會從倒數兩列開始檢查），若大於 0，則從定位原點所在列數開始，此設計可以判斷圖形下落到任何位置時，哪列可以做為定位線停下。

範例說明 以橘色 L 型方塊為例，紅色框框為第 16 列第 4 行的檢查範圍，第 16 列為定位線。

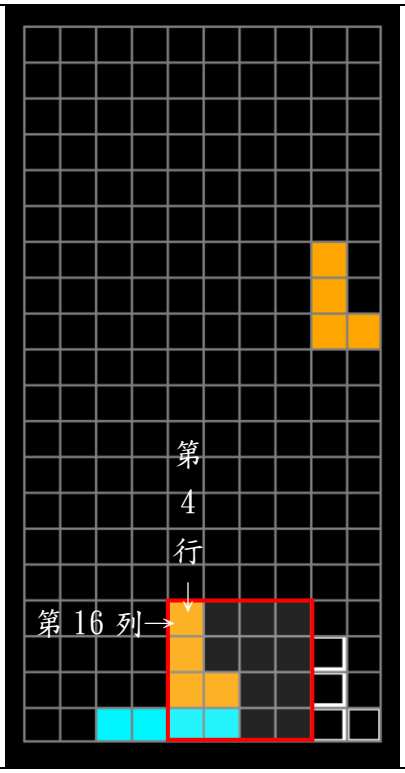

圖 4.29 、定位線檢查示意圖

(資料來源：研究者自行繪製)

因 Pygame 的 draw 函式沒辦法畫一次就一直停留在畫面上，所以我另尋一個辦法，將已停下的方塊存放在一起，最後再繪製在畫面上。

```
76  class Move():
```

```
77      def init(self):
78          self.speed = 0.5
79          self.draw_init()
80          self.lines = []
81          for i in range((HEIGHT-100)//30):
82              self.lines.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
83          for i in range(3):
84              self.lines.append([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
85          self.imgs = []
```

圖 4.30、建立存放停下方塊的序列

程式說明

[85]在 init 函式中建立 imgs 序列存放所有方塊停下後的圖形及其座標。

建立一函式執行讓方塊到定位線時停下或直接下落，並在方塊停下時，將方塊及座標存入剛剛建立的序列。

```
76  class Move():
```

```
158     def go_down(self):
159         for j in range(len(self.n)):
160             self.X, self.Y = position.all_spaces[self.n[j]]
161             self.X += self.SPEEDx
162             self.Y = (self.stop_line+position.mn_dic[self.n[j]][0])*30+50
163             self.image = pygame.Surface((30, 30))
164             self.image.fill(self.color)
165             self.rect = self.image.get_rect()
166             self.rect.x = self.X
167             self.rect.y = self.Y
168             self.imgs.append([self.image, [self.X, self.Y]])
169             self.lines[(self.Y-50)//30][(self.X-100)//30] = 1
170             if j == 3:
171                 self.draw_init()
```

圖 4.31、將方塊及座標加入序列

程式說明

[158-171]建立 go_down 函式，使圖形到達定位線時停下，或是按下空白鍵後直接落下。

[162]獲取應停下的 y 座標值。

[163-168]建立畫布做為已停下的方塊以便後續顯示於畫面上，並將畫布及

其座標加入 imgs 序列。

[169]將方塊新填滿的格子的位置從未填滿（0）更新為已填滿（1）。

[170-171]當四個方塊皆完成，則執行 draw_init 函式，進行下一個初始方塊數值的重置。

為了判斷玩家是否按下空白鍵（執行直接落下），建立變數存放布林值。

```
76   class Move():
122      def draw_init(self):
123          #隨機選擇初始圖形
124          self.choose = random.choice(all_objects[0:7])
125          self.n = random.choice(self.choose)
126          self.color = all_colors[all_objects.index(self.choose)]
127          #定位原點
128          self.O = [-2, 4]
129          #方塊座標y值增加數值
130          self.speed_n = 0
131          self.SPEEDy = 0
132          #方塊座標x值增加數值
133          self.SPEEDx = 0
134          self.Go_Down = False
```

圖 4.32、是否按下空白鍵的變數

程式說明

[134]在 draw_init 函式中加入 Go_Down 變數，存放是否按下空白鍵的布林值。

判斷是否按下空白鍵，並改變其布林值。

```
245  while running:
269      #取得輸入
270      for event in pygame.event.get():
271          if event.type == pygame.QUIT:
272              running = False
273          if event.type == pygame.KEYDOWN and move.gameover == False:
274              #移動系統
275              if event.key == pygame.K_RIGHT:
276                  move.move("R")
277              if event.key == pygame.K_LEFT:
278                  move.move("L")
279              if event.key == pygame.K_SPACE:
280                  move.Go_Down = True
```

圖 4.33、判斷是否按下空白鍵

程式說明

[279-280] 當按下空白鍵時，Go_Down 則會變為 True。

除了按下空白鍵時，當方塊到達定位線時也要執行 go_down 函式。

```
76  class Move():
```

```
102     def draw(self):
103         if (self.speed*self.speed_n) % 30 == 0:
104             self.SPEEDy = (self.speed)*self.speed_n
105             self.O[0] += 1
106         self.region_judge()
107         for j in range(len(self.n)):
108             self.X, self.Y = position.all_spaces[self.n[j]]
109             self.X += self.SPEEDx
110             self.Y += self.SPEEDy
111             self.judge_touch()
112             if self.O[0] >= 0: O = self.O[0]
113             else: O = 0
114             for i in range(O, len(self.judge_list)):
115                 if self.judge_list[i] == False:
116                     self.stop_line = i-1
117                     break
118                 if self.judge_list.count(True) == 19:
119                     self.stop_line = 18
120             if (self.Y-50)//30 >= self.stop_line+position.mn_dic[self.n[j]][0] or self.Go_Down == True:
121                 self.go_down()
122             else:
123                 if self.Y >= 50:
124                     fill_rect = pygame.Rect(self.X, self.Y, 30, 30)
125                     pygame.draw.rect(screen, self.color, fill_rect)
126         self.speed_n += 1
```

圖 4.34、使方塊在定位線停下

程式說明

[120-121]當方塊落下至定位線或按下空白鍵時，執行 go_down 函式。

將停下的所有方塊繪製到畫面上。

```
245 while running:
```

```
286     #畫面顯示
287     screen.fill(BLACK)
288     move.draw()
289     for i, j in move.imgs:
290         screen.blit(i, j)
291     #遊戲畫面顯示
292     outline_rect = pygame.Rect(100, 50, BAR_WIDTH, BAR_HEIGHT)
293     pygame.draw.rect(screen, LIGHT_GRAY, outline_rect, 2)
294     for i in range(9):
295         pygame.draw.line(screen, LIGHT_GRAY, (130+i*30, 50), (130+i*30, 648), 2)
296     for i in range(19):
297         pygame.draw.line(screen, LIGHT_GRAY, (100, 80+i*30), (398, 80+i*30), 2)
298
299     pygame.display.update()
300 pygame.quit()
```

圖 4.35、繪製已停下方塊

程式說明

[289-290]將 imgs 中已停下的方塊繪製到螢幕上。

四、落下預判框線

因自己和同學在測試是否有 bug 時皆發現若沒有預判落下位置的框線的話，很容易發生看錯行導致放錯位置的情形，故我增加了此功能。

```
76    class Move():
```

```
102     def draw(self):
103         if (self.speed*self.speed_n) % 30 == 0:
104             self.SPEEDy = (self.speed)*self.speed_n
105             self.O[0] += 1
106         self.region_judge()
107         for j in range(len(self.n)):
108             self.X, self.Y = position.all_spaces[self.n[j]]
109             self.X += self.SPEEDx
110             self.Y += self.SPEEDy
111             self.judge_touch()
112             if self.O[0] >= 0: O = self.O[0]
113             else: O = 0
114             for i in range(O, len(self.judge_list)):
115                 if self.judge_list[i] == False:
116                     self.stop_line = i-1
117                     break
118                 if self.judge_list.count(True) == 19:
119                     self.stop_line = 18
120             if (self.Y-50)//30 >= self.stop_line+position.mn_dic[self.n[j]][0] or self.Go_Down == True:
121                 self.go_down()
122             else:
123                 if self.Y >= 50:
124                     fill_rect = pygame.Rect(self.X, self.Y, 30, 30)
125                     pygame.draw.rect(screen, self.color, fill_rect)
126                 if self.stop_line >= 0:
127                     outline_rect = pygame.Rect(self.X, (self.stop_line+position.mn_dic[self.n[j]][0])*30+50, 30, 30)
128                     pygame.draw.rect(screen, WHITE, outline_rect, 3)
129         self.speed_n += 1
```

圖 4.36、繪製落下預判框線

程式說明

[157-159]繪製白色框線在方塊應落下的位置。

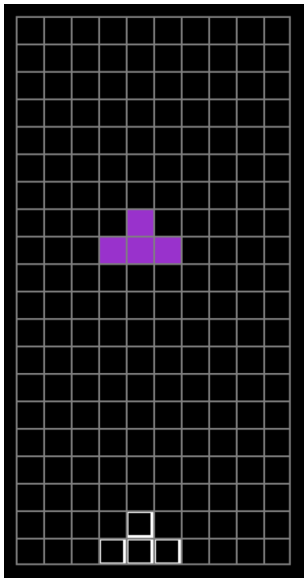

圖 4.37、落下預判框線示意圖

五、判定何時 gameover

為了判斷是否遊戲結束，建立變數存放布林值。

```
76  class Move():
```

```
77      def init(self):
78          self.speed = 0.5
79          self.draw_init()
80          self.lines = []
81          for i in range((HEIGHT-100)//30):
82              self.lines.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
83          for i in range(3):
84              self.lines.append([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
85          self.imgs = []
86          self.gameover = False
```

圖 4.38、是否遊戲結束的變數

程式說明

[86]在 init 函式中加入 gameover 變數，存放是否遊戲結束的布林值。

判斷是否遊戲結束並改變其布林值。

```
76  class Move():
```

```
163     def go_down(self):
164         for j in range(len(self.n)):
165             self.X, self.Y = position.all_spaces[self.n[j]]
166             self.X += self.SPEEDx
167             self.Y = (self.stop_line+position.mn_dic[self.n[j]][0])*30+50
168             self.image = pygame.Surface((30, 30))
169             self.image.fill(self.color)
170             self.rect = self.image.get_rect()
171             if self.Y >= 50:
172                 self.rect.x = self.X
173                 self.rect.y = self.Y
174                 self.imgs.append([self.image, [self.X, self.Y]])
175                 self.lines[(self.Y-50)//30][(self.X-100)//30] = 1
176                 if j == 3:
177                     self.draw_init()
178                 if self.Y == 50:
179                     self.gameover = True
180             if self.Y < 50:
181                 self.gameover = True
```

圖 4.39、判斷是否遊戲結束

程式說明

[178-181]在 go_down 函式中加入判斷式，若停下後的方塊碰到或超出遊戲邊界，則 gameover 變為 True。而若方塊已超出遊戲邊界，則不繪製。

在測試時，發現在遊戲結束後，仍可進行左右移動及直接落下等操作，為了防止這個狀況，故設定為在遊戲結束後玩家不能再進行方塊的操作。

```
245   while running:
```

```
271       #取得輸入
272       for event in pygame.event.get():
273           if event.type == pygame.QUIT:
274               running = False
275           if event.type == pygame.KEYDOWN and move.gameover == False:
276               #移動系統
277               if event.key == pygame.K_RIGHT:
278                   move.move("R")
279               if event.key == pygame.K_LEFT:
280                   move.move("L")
281               if event.key == pygame.K_SPACE:
282                   move.Go_Down = True
```

圖 4.40、遊戲結束時禁止執行方塊操作

程式說明

[275]在遊戲主迴圈取得輸入的判斷式加入新條件，則若遊戲結束，玩家無法再移動方塊。

4.4 旋轉系統

建立一個 Rotate 類別存放所有旋轉系統的屬性及函式並建立一函式執行方塊旋轉。

```
182  class Rotate():
```

```
184      def rotate(self):
185          d = move.choose.index(move.n)+1
186          if d-len(move.choose) >= 0:
187              d = d-len(move.choose)
188          move.n = move.choose[d]
189          move.draw()
```

```
233  rotate = Rotate()
```

圖 4.41、方塊旋轉函式

程式說明

[184-189]建立 rotate 函式，當按上鍵時，圖形向右轉。

[185]因各圖形旋轉態的序列皆按照向右轉的順序排列，故只要取下一個序列即可取得向右旋轉後的方塊編號。

[189]取得新的方塊編號後，重新繪製圖形。

[233]將變數 rotate 加入 Rotate 類別

判斷是否執行方塊旋轉函式（是否按下上鍵）。

```
245  while running:
```

```
271      #取得輸入
272      for event in pygame.event.get():
273          if event.type == pygame.QUIT:
274              running = False
275          if event.type == pygame.KEYDOWN and move.gameover == False:
276              #移動系統
277              if event.key == pygame.K_RIGHT:
278                  move.move("R")
279              if event.key == pygame.K_LEFT:
280                  move.move("L")
281              if event.key == pygame.K_SPACE:
282                  move.Go_Down = True
283              if event.key == pygame.K_UP:
284                  rotate.rotate()
```

圖 4.42、判斷是否按下上鍵

程式說明

[283-284]若按上鍵，則執行 rotate 函式，將圖形向右旋轉。

4.5 消除系統

一、 判斷可消除列

建立一個 Break 類別存放所有消除系統的屬性及函式並建立一函式檢查是否有層列全被方塊填滿。

```
191  class Break():
```

```
197      def break_judge(self):
198          for i in move.lines:
199              judge_filled = 0
200              for j in i:
201                  if j == 1 and move.lines.index(i)<=19:
202                      judge_filled += 1
203                  else:break
204              if judge_filled == 10:
205                  move.lines.remove(i)
206                  move.lines.insert(0, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
234  break_line = Break()
```

圖 4.43、檢查層列是否被方塊填滿

程式說明

[197-206]建立 break_judge 函式判斷是否有可消除的列（皆被方塊填滿）。

[200-203]透過迴圈檢查每一列每一格是否皆被填滿（數值皆為 1）。

[204-206]若 10 格皆被填滿，則將此列刪除並加入新的一列（設為皆未被填滿）。

[234]將變數 break_line 加入 Break 類別。

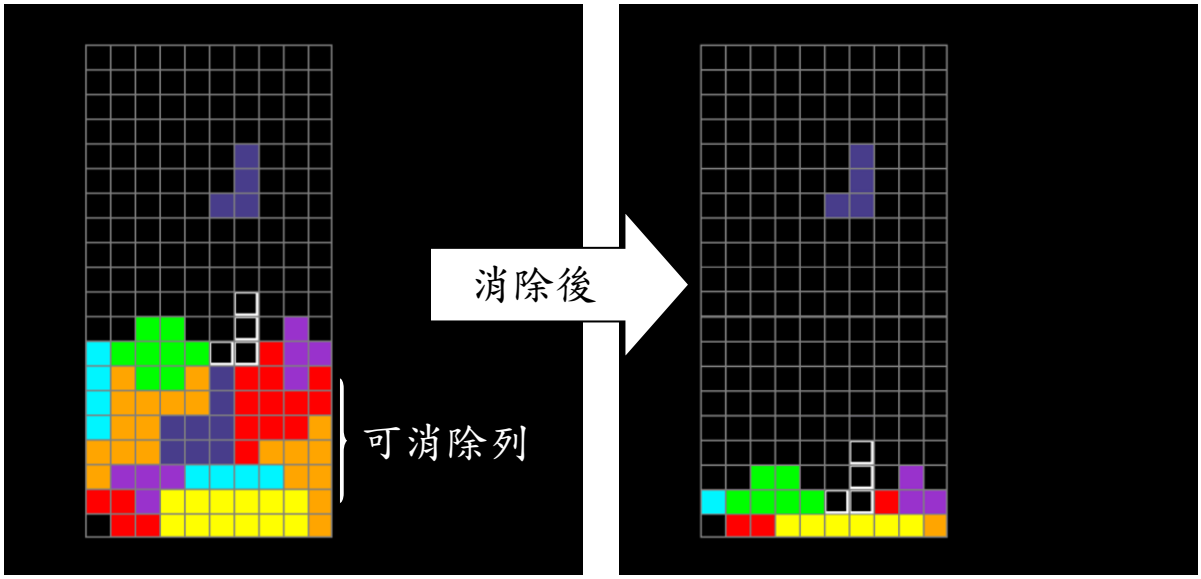

圖 4.44、可消除列消除前後示意圖

二、 刪除可消除列並更新圖形及分數

建立分數初始值存放玩家分數。

```
191  class Break():
```

```
192      def init(self):
193          self.score = 0
```

```
236  break_line.init()
```

圖 4.45、建立分數初始值

程式說明

[192-193]建立 init 函式並存入分數初始值。

[236]執行 init 函式重置分數。

建立一函式來消除可消除列並增加分數。

```
191  class Break():
```

```
210      def break_lines(self, line):
211          self.imgs_re = []
212          self.score += int(move.speed*10)
213          for h in range(len(move.imgs)):
214              if move.imgs[h][1][1] == line*30+50:
215                  self.imgs_re.append(move.imgs[h])
216              elif move.imgs[h][1][1] < line*30+50:
217                  move.imgs[h][1][1] += 30
218          for i in self.imgs_re:
219              move.imgs.remove(i)
```

圖 4.46、消除可消除列函式

程式說明

[210-219]建立 break_lines 函式來刪除被消除的圖形並將上層圖形皆下降一格。

[211]建立 imgs_re 序列存放需刪除的方塊　[212]每消除一列，就增加分數。

[214-215]將須刪除的方塊存入序列。

[216-217]將消除列上層方塊下降一格（座標加 30）。

[218-219]將 imgs 內需消除方塊刪除。

執行消除可消除列函式。

```
191  class Break():
```

```
198      def break_judge(self):
199          for i in move.lines:
200              judge_filled = 0
201              for j in i:
202                  if j == 1 and move.lines.index(i)<=19:
203                      judge_filled += 1
204                  else:break
205              if judge_filled == 10:
206                  self.break_lines(move.lines.index(i))
207                  move.lines.remove(i)
208                  move.lines.insert(0, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

圖 4.47、消除可消除列

程式說明

[206]在 break_judge 函式中執行 break_lines 函式，則每發現一次可消除列就刪除。

三、 等級提升

為了增加遊戲豐富度，方塊落下速度要逐漸增快，建立變數存放速度等級。

```
191  class Break():
```

```
192      def init(self):
193          self.score = 0
194          self.score_old = 0
195          self.level = 1
```

圖 4.48、遊戲等級及升等時分數的變數

程式說明

[194-195]隨著分數增加，遊戲等級也會越來越高（速度越來越快），故在 init 函式中加入變數存放上一次升等時的分數及等級數。

當分數增加一個固定的值後，等級隨之提升。

```
245  while running:
```

```
292      #消行系統
293      break_line.break_judge()
294      if break_line.score - break_line.score_old >= 80:
295          break_line.score_old = break_line.score
296          move.speed += 0.25
297          break_line.level += 1
```

圖 4.49、判斷分數增加值

程式說明

[293]在遊戲主迴圈內執行 break_judge 函式來檢查是否有可消除列。

[294-297]每當分數增加 80，則等級上升且速度加快。

4.6 畫面顯示

一、 初始畫面

初始畫面須顯示遊戲遊玩方法，為了判斷是否為第一次開始遊戲，建立一變數存放是否為第一次遊戲的布林值。

```
61    init = True
```

圖 4.50、是否為第一次遊戲的變數

程式說明

[61]建立 init 變數存放是否為第一次開始遊戲。

建立一函式進行文字繪製。

```
224   WHITE = (255, 255, 255)
225   def draw_text(text, size, x, y):
226       font_name = os.path.join("font.ttf")
227       font = pygame.font.Font(font_name, size)
228       #繪製文字(文字, 平滑值, 文字顏色, 背景顏色)
229       TEXT = font.render(text, True, WHITE)
230       TEXT_rect = TEXT.get_rect()
231       TEXT_rect.centerx = x
232       TEXT_rect.centery = y
233       screen.blit(TEXT, TEXT_rect)
```

圖 4.51、繪製文字函式

程式說明

[224-233]建立 draw_text 函式以執行文字繪製。

繪製遊戲初始畫面。

```
245   while running:
244       while init == True:
245           draw_text("TETRIS", 64, WIDTH//2, HEIGHT//2-100)
246           draw_text("左右鍵移動 空白鍵直接落下 上鍵旋轉方塊", 32, WIDTH//2, HEIGHT//2)
247           draw_text("按下任意鍵開始遊戲", 32, WIDTH//2, HEIGHT//2+100)
248           pygame.display.update()
249           for event in pygame.event.get():
250               if event.type == pygame.QUIT:
251                   running = False
252                   init = False
253                   break
254               if event.type == pygame.KEYUP:
255                   init = False
256                   break
```

圖 4.52、繪製遊戲初始畫面

程式說明

[244-256]當為第一次開始遊戲時，進行初始畫面繪製。

[245-248]執行 draw_text 函式以繪製遊戲遊玩方式的說明文字並更新顯示畫面。

[249-256]當按下關閉視窗時則退出此迴圈並將遊戲主迴圈關閉，若按下任意鍵則退出此迴圈（繼續遊戲主迴圈）。

圖 4.53、遊戲初始畫面

二、 遊戲結束畫面

繪製遊戲結束畫面。

```
239   img = pygame.image.load(os.path.join("T_imgs", "level_1.jpg")).convert()
240   gameover_img = pygame.transform.scale(img, (200, 200))
```

```
245   while running:
```

```
267       while move.gameover == True:
268           screen.blit(gameover_img, (550, 200))
269           draw_text("GAME OVER", 72, 650, 460)
270           draw_text("按下任意鍵繼續遊戲", 32, 650, 550)
271           pygame.display.update()
272           for event in pygame.event.get():
273               if event.type == pygame.QUIT:
274                   running = False
275                   move.gameover = False
276               if event.type == pygame.KEYDOWN:
277                   move.init()
278                   break_line.init()
279                   move.gameover = False
```

圖 4.54、繪製遊戲結束畫面

程式說明

[239-240]將遊戲結束時要顯示的圖片載入。

[267-279]當遊戲結束時，進行遊戲結束畫面繪製。

[268-271]進行說明文字及圖片繪製並更新遊戲畫面。

[272-279] 當按下關閉視窗時則退出此迴圈並將遊戲主迴圈關閉，若按下任意鍵則執行移動系統及消行系統的 init 函式重置初始值並退出此迴圈（繼續遊戲主迴圈）。

圖 4.55、遊戲結束畫面

三、 分數及等級

繪製分數及等級。

```
245   while running:
```

```
299       draw_text(f"Score {break_line.score}", 32, 650, 100)
300       draw_text(f"Level {break_line.level}", 32, 650, 150)
```

圖 4.56、繪製分數及等級

程式說明

[299-300]顯示分數和等級的文字及數值。

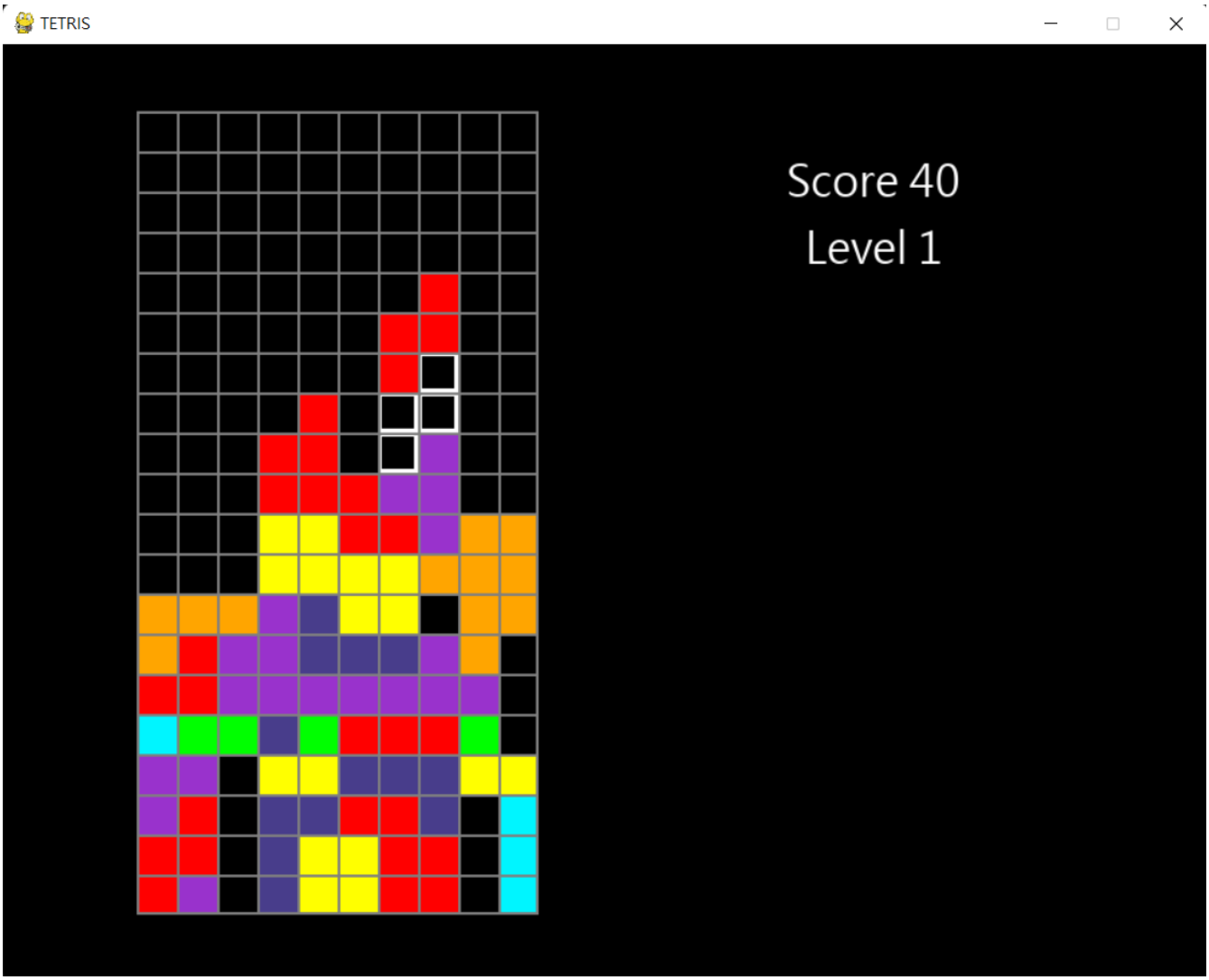

圖 4.57、分數及等級顯示

4.7 音效

一、 混音器初始化

在使用音效之前，須先進行混音器初始化。

```
30    pygame.mixer.init()
```

圖 4.58、混音器初始化

二、 背景音樂

加入遊戲背景音樂。

```
242   pygame.mixer.music.load(os.path.join("T_sounds", "happytime.mp3"))
243   pygame.mixer.music.play(-1)
244   pygame.mixer.music.set_volume(0.05)
```

圖 4.59、遊戲背景音樂

程式說明

[242]載入背景音樂檔案。

[243]將播放音樂函式參數值設為－1（循環播放）。

[244]設定背景音樂的音量大小。

三、得分音效

加入得分音效。

```
246   get_score_sound = pygame.mixer.Sound(os.path.join("T_sounds", "score.mp3"))
```

```
191   class Break():
```

```
208       def break_lines(self, line):
209           self.imgs_re = []
210           self.score += int(move.speed*10)
211           get_score_sound.play()
212           for h in range(len(move.imgs)):
213               if move.imgs[h][1][1] == line*30+50:
214                   self.imgs_re.append(move.imgs[h])
215               elif move.imgs[h][1][1] < line*30+50:
216                   move.imgs[h][1][1] += 30
217           for i in self.imgs_re:
218               move.imgs.remove(i)
```

圖 4.60、得分音效

程式說明

[246]載入得分音效檔案。

[211]得分時播放得分音效。

4.8 總體設計概念

我將想完成的遊戲功能分為 4 個系統，起初我並沒有使用類別來分類不同系統的函式，而造成的結果就是程式碼雜亂無章，每次要修改時都要花很多時間找出問題所在處。因此我後來將這 4 個系統所須函式分別存入不同類別，使用類別的好處有以下兩點：

一、 程式碼更方便閱讀

以往過去的程式設計經驗都僅限於上課練習題目或競賽內容，而這類型的程式碼幾乎都較簡短，不像一個遊戲，需要很多環節一起配合。若是沒使用類別，將會造成程式碼混雜，不只修改不易，也會讓其他人看不懂。尤其是多人一起完成的專案，分類清楚的程式碼是非常重要的，不同系統分成不同類別就可以讓人一眼就清楚明瞭這些函式的功用。

二、 可以快速、大量生產

在之後如果碰到需要類似的程式碼的時候，若我想另開一個檔案製作俄羅斯方塊進階版，能夠使用類別的繼承，就不須在新檔案上貼重複的內容，可以快速且大量的產出類似的程式碼。

第五章 結論與建議

5.1 結論

我成功利用 Python 設計出俄羅斯方塊，並且邀請同學進行遊戲的測試，修正之後，完成了此次的成果，未來我也會再加入一些我想新增的功能使遊戲更完整及好玩。

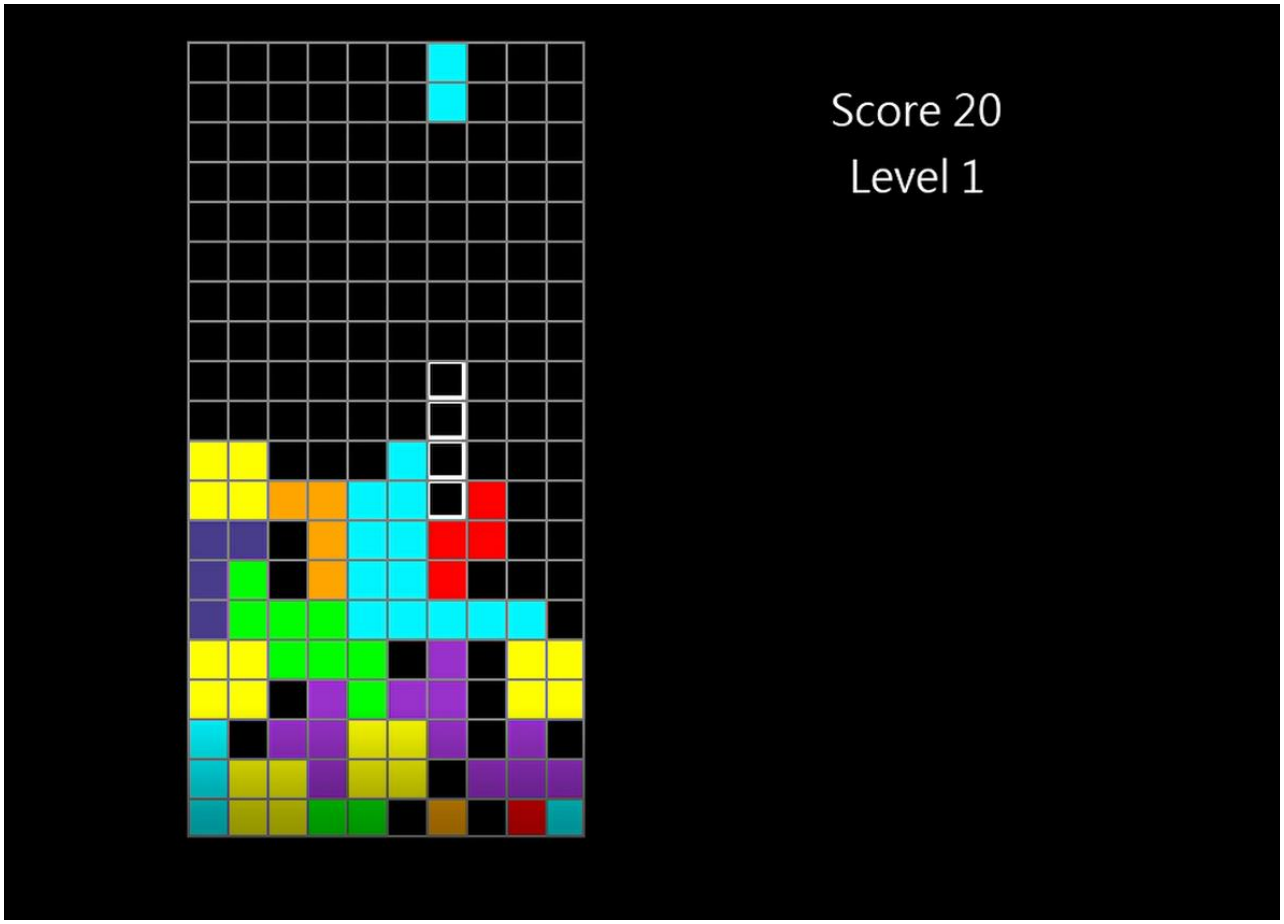

圖 5.1、程式實際運行影片截圖（00:00:35）

影片網址： https://youtu.be/hz0Yj-E1VoQ

5.2 建議

一、增加遊戲豐富度

(一) 遊戲區域形狀改變

將遊戲區域形狀改成特殊形狀（星型等）並將方塊改成對應的圖形。

更改方向 定位方法須重新設定，且方式會比目前複雜。

(二) 方塊種類增加

將方塊改成由五塊小正方形甚至更多塊所組成。

更改方向 可延續使用目前的定位方法，且方塊圖型會更多種。

(三) 增加道具

消除層列時有機率獲得道具。

更改方向 須新增道具動畫以及道具效果的函式。

(四) 增加炸彈

消除層列時有機率出現炸彈（取代某格方塊），若放置方塊時觸碰炸彈則爆炸且一次可消較多層方塊。

更改方向 須新增爆炸動畫以及隨機出現的函式並配合消除系統。

(五) 對戰功能

雙人單機對戰（非線上）。

更改方向 須分成兩個座標系統，其他系統只須將兩個玩家的操作分隔開來，舉例而言，若玩家 A 按了右鍵，則就只能玩家 A 區塊的方塊執行右移函式。

二、Sprite 模組

Pygame 中有 Sprite 模組，其中有可以偵測 sprite 物件是否碰撞的函式，之後可以嘗試使用這個功能來判定方塊是否落地，程式碼或許會比現在的消除系統來得精簡許多，因為我所寫的判斷是否落地的函式是使用偏土法煉鋼的方法，用迴圈一格一格檢查，所以過程較為繁複。

資料來源

1. CSDN 博客-專業 IT 技術發表平台－Pygame 入門教程（一）初始化和主循環
https://blog.csdn.net/baidu_36499789/article/details/113462828#:~:text=%E5%88%9D%E5%A7%8B%E5%8C%96pygame%E5%BA%93%20pygame.init%20%28%29%20%E6%98%AF%E5%9C%A8%E4%BD%BF%E7%94%A8%20pygame%20%E5%BA%93%E4%B9%8B%E5%89%8D%E5%88%9D%E5%A7%8B%E5%8C%96%20pygame,%E5%BA%93%E3%80%82%20init%20%E8%BF%99%E4%B8%AA%E5%8D%95%E8%AF%8D%E7%BC%A9%E5%86%99%E8%A6%81%E8%AE%B0%E5%A5%BD%E4%BA%86%EF%BC%8C%E5%AE%83%E6%98%AF%20initialize%20%28%E5%88%9D%E5%A7%8B%E5%8C%96%29%20%E7%9A%84%E7%BC%A9%E5%86%99%EF%BC%8C%E5%9C%A8%E7%BC%96%E7%A8%8B%E4%B8%AD%E7%BB%8F%E5%B8%B8%E5%87%BA%E7%8E%B0%E3%80%82%202.
2. CSDN 博客-專業 IT 技術發表平台－Python Pygame 常用操作
https://blog.csdn.net/doasmaster/article/details/104364993?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_default&utm_relevant_index=2