期末專題 In a tight corner

資工二 A 107502520 黃允誠

首先，還是用中文再申明一次：我的程式需要 CMD 視窗按照設定。
我有試著找過組語中有沒有指令可以直接動態改，但找很久了也沒看到，就跟之前的”ReadKey”一樣，或許有，但沒辦法資料太少了因為組語已經過氣……不是，我去 irvine32 的整個文檔搜尋過了，沒有。
所以需求的設定是這樣：
點陣字型，大小改 8x8，然後視窗跟 BUFFER 大小都改 100x100。
沒辦法只好請助教們跟教授動手改了。

接著另一個申明，請助教們跟教授開適當的 IDE 來讀程式碼。如果用 lab 課程中的方式(AKA 記事本)，那就是說看不起我們，認為大家的程式跟 lab 課的小練習一樣複雜，那看不懂就不是我們的問題了……

接著要講一下，玩遊戲很爽，寫遊戲很痛苦。雖然遊戲是所有程式當中最沒用的，但絕對是最難寫的，所以不要看不起遊戲程式……特別是”即時制”的，再加上有碰撞、物理相關的遊戲，都會很糟糕。
在寫這種程式有個要點，就是要進行大量的模組化，把所有功能分開，可以做一次的事情，就只做一次。
所以從我的程式碼可以看到，700 行的程式，有 100 行都是 macro，因為在組語中，能做到這件事情的，macro 是最接近的。
這並不是說程式碼多了 100 行，而是因為這 100 行當中做的事情，每次都要好幾行，而且整個程式各處要做好幾遍，要是把這 100 行展開，我的程式就再也沒人看得懂了(雖然現在也可能只有我看得懂，但至少有一個人……)

除了 macro 之外，底下也是全部弄成一堆 procedure。雖然後來我發現，頻繁的 invoke ”好像”會降低程式效能，會跑比較慢，但是這也沒辦法，因為易讀性跟效能常常就站在兩邊。我個人永遠選擇易讀性，反正現在電腦很猛……

然後，這個小遊戲的靈感我報告的時候有提到了：ACG 作品中常出現的，巨石群間墜落的動作場面。這個英文名稱，是”陷入絕境”的意思。
接下來，我再把遊戲規則貼過來一下：
隕石會隨機出現，隕石間相撞會崩壞。
方向鍵控制主角移動，每次移動最多轉彎一次。
撞上隕石移動方向的正面或碰到邊界，遊戲結束。
碰到隕石側面或背後，可以黏附上隕石並重製轉彎次數限制。
黏附的隕石崩壞時，主角進入漂浮狀態，緩慢按照慣性移動。

最後，在程式碼當中，”refresh”的部分，我特別註解建議助教們跟教授跳過，因為實在太亂了。雖然我對自己的排版還算有信心，而且每個區塊我都有給 Label，就算是我自己寫的迴圈 macro 也支援給每個迴圈名字，我也確實盡量給了符合要實作功能的名字。其實我認為，要是很認真地埋頭讀應該也許可能可以稍微看懂一點點，但是這樣對助教們和老師太過分了……所以，以下我會一塊一塊的，把 refresh 這個部分，就是所有資料更新的地方解釋完。
解釋完之後，我自己可能也會瀕臨崩潰，所以現在先說這句話：報告到這邊結束。

Refresh PROC

每過一段時間(boulderCreateInterval)，程式就會刷新出一個新的隕石。
我的實作方法是：從所有隕石的 data 中，隨便找一個沒在場上的，然後隨機初始化所有參數，最後放到場上。
因此雖然是一個陣列，但所有隕石之間是獨立的，沒有先後次序。每個隕石只做自己的事情，偵測跟自己有關的事件，記錄自己的狀態。這也是模組化的一種表現，物件導向。

```
CreateBoulder:
    TIMER boulderCreationTimer, boulderCreateInterval
        newBoulder TEXTEQU <(OBJECT PTR boulders[esi])>
        FORLOOP FindBoulder, ecx, boulderNum, esi, TYPE OBJECT
            mov bl, newBoulder.status
            .IF bl == statusNotOnField
                BREAK FindBoulder
            .ENDIF
        ENDFL FindBoulder, ecx, boulderNum, esi, TYPE OBJECT
        mov newBoulder.style, 0
        vmov newBoulder.operationsTimer, boulderCreationTimer
        rand newBoulder.objSize.x, boulderMinW, boulderMaxW
        rand newBoulder.objSize.y, boulderMinH, boulderMaxH
        rand newBoulder.weight, boulderMinWeight, boulderMaxWeight
        rand bx, 0, 3
        .IF bx == 0
            mov newBoulder.direction.x, 0
            mov newBoulder.direction.y, -1
            rand newBoulder.position.x, 0, fieldW
            mov newBoulder.position.y, (fieldH-1)
            vadd newBoulder.position.y, newBoulder.objSize.y
        .ELSEIF bx == 1
            mov newBoulder.direction.x, 0
            mov newBoulder.direction.y, 1
            rand newBoulder.position.x, 0, fieldW
            mov newBoulder.position.y, 0
            vsub newBoulder.position.y, newBoulder.objSize.y
        .ELSEIF bx == 2
```

```
                mov newBoulder.direction.x, -1
                mov newBoulder.direction.y, 0
                mov newBoulder.position.x, (fieldW-1)
                vadd newBoulder.position.x, newBoulder.objSize.x
                rand newBoulder.position.y, 0, fieldH
            .ELSEIF bx == 3
                mov newBoulder.direction.x, 1
                mov newBoulder.direction.y, 0
                mov newBoulder.position.x, 0
                vsub newBoulder.position.x, newBoulder.objSize.x
                rand newBoulder.position.y, 0, fieldH
            .ENDIF
            mov newBoulder.status, statusMoving
        ENDT
```

接下來，對所有在場上的隕石，進行移動。

除了按照隕石本身紀錄的方向跟移動的時間間距來做移動之外，同時也進行所有的碰撞判定：隕石跟隕石之間(崩解，如果玩家黏附著這些隕石其中之一，同時處理玩家進入漂浮狀態)，隕石跟玩家角色之間(遊戲結束，因為隕石一動碰到玩家，那絕對是隕石正面撞上去，不可能是從側面或背後黏附的情況)，隕石跟邊界(超出邊界要移除隕石釋出 data)。

另外，因為按照遊戲規則，玩家可以黏附在隕石上，我實作的方法是：黏附的時候，玩家本身不移動，但黏附的那顆隕石移動的時候，它帶著玩家一起移動。

因此這邊同時也判斷隕石跟玩家間的關係，帶著玩家移動的同時，也要判定玩家跟邊界的碰撞(撞到邊界遊戲結束)

```
    FORLOOP MoveBoulder, ecx, boulderNum, esi, TYPE OBJECT
        moving TEXTEQU <(OBJECT PTR boulders[esi])>
        mov bl, moving.status
        TIMER moving.operationsTimer, moving.weight, bl == statusMoving
            vadd moving.position.x, moving.direction.x
            vadd moving.position.y, moving.direction.y
            .IF player.interactingID == esi && player.status == statusInteracting
                vadd player.position.x, moving.direction.x
                vadd player.position.y, moving.direction.y
                INVOKE CollisionTest, player.position.y, player.position.y, player.position.x,
player.position.x, ft, fb, fl, fr
                .IF ax == 0
                    vsub player.position.x, moving.direction.x
                    vsub player.position.y, moving.direction.y
                    INVOKE GameOver
                .ENDIF
            .ENDIF
            vmov (WORD PTR tmpWORD[0]), moving.position.y
            vsub (WORD PTR tmpWORD[0]), moving.objSize.y
```

```
mt TEXTEQU <(WORD PTR tmpWORD[0])>
vmov (WORD PTR tmpWORD[2]), moving.position.y
vadd (WORD PTR tmpWORD[2]), moving.objSize.y
mb TEXTEQU <(WORD PTR tmpWORD[2])>
vmov (WORD PTR tmpWORD[4]), moving.position.x
vsub (WORD PTR tmpWORD[4]), moving.objSize.x
ml TEXTEQU <(WORD PTR tmpWORD[4])>
vmov (WORD PTR tmpWORD[6]), moving.position.x
vadd (WORD PTR tmpWORD[6]), moving.objSize.x
mr TEXTEQU <(WORD PTR tmpWORD[6])>
```

這邊就是隕石之間的碰撞，上面很大一坨只是變數的前處理。(正在測試碰撞的隕石的四個邊)

```
FORLOOP BoulderToBoulderCollision, edx, boulderNum, edi, TYPE OBJECT
    testing TEXTEQU <(OBJECT PTR boulders[edi])>
    mov bh, testing.status
    .IF edi != esi && bh != statusNotOnField
        vmov (WORD PTR tmpWORD[8]), testing.position.y
        vsub (WORD PTR tmpWORD[8]), testing.objSize.y
        tt TEXTEQU <(WORD PTR tmpWORD[8])>
        vmov (WORD PTR tmpWORD[10]), testing.position.y
        vadd (WORD PTR tmpWORD[10]), testing.objSize.y
        tb TEXTEQU <(WORD PTR tmpWORD[10])>
        vmov (WORD PTR tmpWORD[12]), testing.position.x
        vsub (WORD PTR tmpWORD[12]), testing.objSize.x
        tl TEXTEQU <(WORD PTR tmpWORD[12])>
        vmov (WORD PTR tmpWORD[14]), testing.position.x
        vadd (WORD PTR tmpWORD[14]), testing.objSize.x
        tr TEXTEQU <(WORD PTR tmpWORD[14])>
```

這邊上面很大一坨也是變數的前處理。(測試碰撞的對象隕石的四個邊)

```
        INVOKE CollisionTest, mt, mb, ml, mr, tt, tb, tl, tr
        .IF ax == 1
            mov moving.status, statusCollapsing
            .IF player.interactingID == esi && player.status == statusInteracting
                vmov player.direction.x, moving.direction.x
                vmov player.direction.y, moving.direction.y
                mov player.moveInterval, playerFloatInterval
                INVOKE GetTickCount
                mov player.operationsTimer, eax
                mov player.status, statusFloating
            .ENDIF
            .IF bh != statusCollapsing
                mov testing.status, statusCollapsing
                .IF player.interactingID == edi && player.status == statusInteracting
```

```
                                vmov player.direction.x, testing.direction.x
                                vmov player.direction.y, testing.direction.y
                                mov player.moveInterval, playerFloatInterval
                                INVOKE GetTickCount
                                mov player.operationsTimer, eax
                                mov player.status, statusFloating
                            .ENDIF
                        .ENDIF
                        BREAK BoulderToBoulderCollision
                    .ENDIF
                .ENDIF
            ENDFL BoulderToBoulderCollision, edx, boulderNum, edi, TYPE OBJECT
```

這邊就接著針對玩家的碰撞。

```
            BoulderToPlayerCollision:
                INVOKE CollisionTest, mt, mb, ml, mr, player.position.y, player.position.y,
player.position.x, player.position.x
                .IF ax == 1
                    vadd player.position.x, moving.direction.x
                    vadd player.position.y, moving.direction.y
                    INVOKE GameOver
                .ENDIF
```

最後就是超過邊界的話把它移除釋出 data，之後要生成新的隕石要用。

```
            BoulderRemove:
                INVOKE CollisionTest, mt, mb, ml, mr, ft, fb, fl, fr
                .IF ax == 0
                    mov moving.status, statusNotOnField
                .ENDIF
        ENDT
    ENDFL MoveBoulder, ecx, boulderNum, esi, TYPE OBJECT
```

好了隕石終於全部移完了，接下來移玩家。

一樣，要進行玩家對隕石的判定，這邊就要看方向決定是遊戲結束還是黏附上去。

```
    MovePlayer:
        TIMER player.operationsTimer, player.moveInterval, player.status == statusFloating ||
player.status == statusMoving || player.status == statusTurned
            vadd player.position.x, player.direction.x
            vadd player.position.y, player.direction.y
            FORLOOP PlayerToBoulderCollision, ecx, boulderNum, esi, TYPE OBJECT
                testing TEXTEQU <(OBJECT PTR boulders[esi])>
                mov bl, testing.status
                .IF bl == statusMoving
                    vmov (WORD PTR tmpWORD[0]), testing.position.y
                    vsub (WORD PTR tmpWORD[0]), testing.objSize.y
```

```
                    tt TEXTEQU <(WORD PTR tmpWORD[0])>
                    vmov (WORD PTR tmpWORD[2]), testing.position.y
                    vadd (WORD PTR tmpWORD[2]), testing.objSize.y
                    tb TEXTEQU <(WORD PTR tmpWORD[2])>
                    vmov (WORD PTR tmpWORD[4]), testing.position.x
                    vsub (WORD PTR tmpWORD[4]), testing.objSize.x
                    tl TEXTEQU <(WORD PTR tmpWORD[4])>
                    vmov (WORD PTR tmpWORD[6]), testing.position.x
                    vadd (WORD PTR tmpWORD[6]), testing.objSize.x
                    tr TEXTEQU <(WORD PTR tmpWORD[6])>
                    INVOKE CollisionTest, player.position.y, player.position.y, player.position.x,
player.position.x, tt, tb, tl, tr
                    .IF ax == 1
                        vsub player.position.x, player.direction.x
                        vsub player.position.y, player.direction.y
                        vmov (WORD PTR tmpWORD[8]), player.direction.x
                        vadd (WORD PTR tmpWORD[8]), testing.direction.x
                        vmov (WORD PTR tmpWORD[10]), player.direction.y
                        vadd (WORD PTR tmpWORD[10]), testing.direction.y
                        .IF (WORD PTR tmpWORD[8]) == 0 && (WORD PTR tmpWORD[10]) == 0
                            INVOKE GameOver
                        .ELSE
                            mov player.status, statusInteracting
                            mov player.interactingID, esi
                        .ENDIF
                        BREAK PlayerToBoulderCollision
                    .ENDIF
                .ENDIF
            ENDFL PlayerToBoulderCollision, ecx, boulderNum, esi, TYPE OBJECT
            INVOKE CollisionTest, player.position.y, player.position.y, player.position.x, player.position.x,
ft, fb, fl, fr
            .IF ax == 0
                vsub player.position.x, player.direction.x
                vsub player.position.y, player.direction.y
                INVOKE GameOver
            .ENDIF
        ENDT
```

接下來，是隕石的崩解動畫。

我是用狀態的方式記錄，除了平常一般的各種狀態之外，另外有個”造型”狀態，在開始崩解的時候，透過這個”造型編號”改變繪製用的字元，像影格一樣達到動畫的效果。

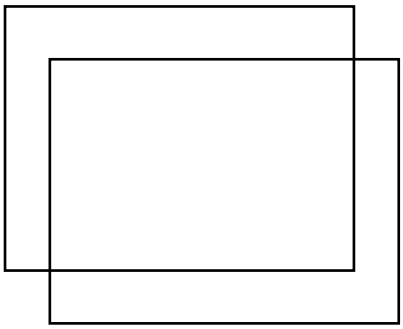
玩家的崩解，因為同時也是死亡動畫，只在遊戲結束時才會有，因此是寫在遊戲結束的那個區塊。

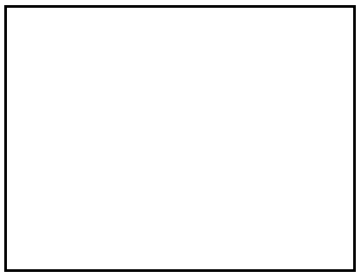
```
    FORLOOP BoulderCollapse, ecx, boulderNum, esi, TYPE OBJECT
```

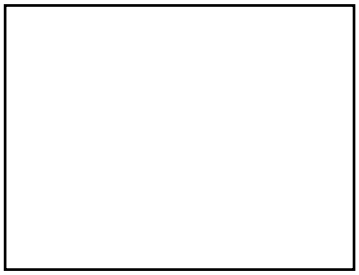
```
        collapsing TEXTEQU <(OBJECT PTR boulders[esi])>
        mov bl, collapsing.status
        TIMER collapsing.operationsTimer, collapseAnimationInterval, bl == statusCollapsing
            mov bl, collapsing.style
            .IF bl >= 5
                mov collapsing.status, statusNotOnField
            .ELSE
                inc collapsing.style
            .ENDIF
        ENDT
    ENDFL BoulderCollapse, ecx, boulderNum, esi, TYPE OBJECT

    RET

Refresh ENDP
```

這邊結束……其他部分，一方面區塊很小，比較簡單，而且我都有註解，沒有像這個這麼複雜，因此就不再多做贅述。很幸運的，我現在還很清醒，這就是寫程式模組化的好處……

最後順便說一下碰撞判定，那個 CollisionTest 是這樣的：
基本上就是把兩個方框的四個邊都丟進去，垂直邊有水平座標，水平邊有垂直座標，這樣總共八個邊，然後進行比較。







如果是有碰撞(像左邊)，那你一定有這個狀況：A 矩形的最左邊要在 B 矩形最右的左邊，同時 A 矩形的最右邊要在 B 矩形最左的右邊。這個只是水平軸的方面，垂直軸同理。同時成立的話，就是有重疊，反之則是分開的，向右邊水平軸就不成立。
所以就是八個邊丟進來，全部比一比就可以了。說是這樣，但每次碰撞判定都來一遍就瘋了，所以另外寫。

大概就這樣，助教們跟教授都辛苦了，以上。

至於 DEMO 的截圖，我個人是認為我第二個禮拜的 DEMO 應該是很清楚了，如果教授認為沒有的話……痾，至少我期中考考得還不錯……我絕對不會說我是已經累到不知道要截圖甚麼……