

ASM Final Project

(A) 主程式說明

引入函式庫 Irvine32.inc，設定螢幕上每個方格的大小如下(一張圖是邊長為 8 個方格的正方形):

```
BLOCK_WIDTH = 6
BLOCK_HEIGHT = 3
MAP_WIDTH = BLOCK_WIDTH * 8
MAP_HEIGHT = BLOCK_HEIGHT * 8
```

以及總關卡數 LEVELS = 3。每一關卡的資料結構含 8*8 的二維陣列和汽車的陣列，如下:

地圖(N 牆壁, Y 空格, Z 出口)	汽車陣列
L1_graph byte "NNNNNNNN", "NBBCYYN", "NYCYDDN", "NAAYYKGZ", "NEYHHKGN", "NEYIYJJN", "NFFIYYN", "NNNNNNNN", 0	L1_CarArr Car <"A", "H", 2, <2, 3>, <1, 3>, 1, 0C1h> , <"B", "H", 2, <2, 1>, <1, 1>, 0, 64h> ,

一輛汽車是一個 STRUCT

```
Car STRUCT
    what BYTE ? ;編號ABC...
    orient BYTE ? ;方向"H"或"Z"
    len BYTE ? ;車子長度
    head COORD <?, ?> ;車頭
    tail COORD <?, ?> ;車尾
    target BYTE ? ;選擇此車輛? "1"代表是, "0"代表否
    color WORD ? ;車子顏色
Car ENDS
```

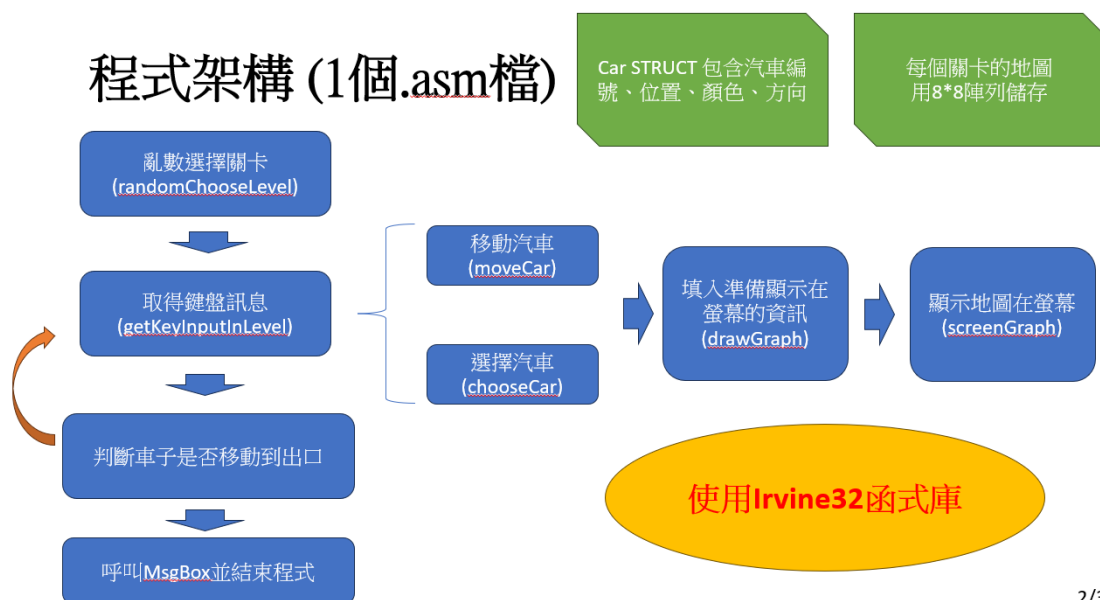
方便計算，以最右端或最下方的車頭為 head，最左邊或最上面為 tail。

主程式如下:

- (1) 呼叫 GetStdHandle，準備螢幕輸出環境
- (2) 隨機選關，呼叫 randomChooseLevel，這裡用到 Irvine 內提供的 Randomize 和 RandomRange 功能，使每次能產生不一樣的亂數，範圍在關卡總數量之間，最後根據亂數值挑選關卡(使用 jump 指令)
- (3) 每個關卡在一個標籤(label)內，呼叫 getKeyInputInLevel，把這一關卡的地圖陣列和汽車陣列作為參數傳入，getKeyInputInLevel 裡面自行呼叫移動

車輛等邏輯判斷的程式，等一下說明。而 `getKeyInputInLevel` 在 `return` 之後即可判斷目標紅車是否已經移動到出口，放在 `ebx` 裡面，若為 0 代表尚未到出口，`jump` 指令跳回再執行 `getKeyInputInLevel` 一次，若為 1 則玩家獲勝，`jump` 指令跳到遊戲結束的標籤。

(4) 遊戲結束標籤(FINISH_GAME)內，呼叫 `MsgBox` 及 `ClsScr`，程式執行完畢。



2/3

(B) 函數說明

(1) 函式 `getKeyInputInLevel` 傳入參數有

```

graph: PTR BYTE, ;地圖陣列
arr: PTR Car, ;汽車陣列
arrSize: DWORD ;汽車陣列長度(汽車數量)
    
```

主體架構如下：

```

INVOKE drawGraph, graph, arr, arrSize
INVOKE screenGraph
mov ebx, 0 ;判斷車子是否移到出口
call ReadChar
    
```

接下來判斷鍵盤訊息：

```

; .IF ax > 4000h ;移動車子，方向鍵
    INVOKE moveCar, graph, arr, arrSize, ax
    ; .IF eax==1 ;函式moveCar會判斷車子是否移動到出口，若是則eax=1，否則eax=0
        mov ebx, 1 ;把eax參數傳遞至ebx
        jmp Finish_Input
    ; .ENDIF
; .ELSE ;選擇車輛，編號ABC...
    
```

```

        INVOKE chooseCar, arr, arrSize, ax
    .ENDIF
Finish_Input:
    ret

```

(2)函式 moveCar 主體架構如下:

```

mov eax, 0 ;判斷車子移動到出口的暫存器
mov ecx, arrSize
mov esi, arr
mov edi, graph
Ll_moveCar: ;尋找選擇的是哪一輛車
    cmp (Car PTR [esi]).target, 1
    je GOT_CAR
    add esi, TYPE Car
    loop Ll_moveCar
GOT_CAR: ;所選的車子，它在汽車陣列中的索引，已經找到
    cmp (Car PTR [esi]).orient, "H" ;汽車有水平和垂直方向區別
    je MOVE_HORIZON
    jne MOVE_VERTICAL

```

接著(舉水平方向移動為例):

```

MOVE_HORIZON:
    mov ebx, 0
    mov dl, (Car PTR [esi]).what
    .IF key==4B00h ;向左移動
        mov bx, (Car PTR [esi]).tail.Y ;車子左端是tail，所以不能用head
        imul bx, 8
        add bx, (Car PTR [esi]).tail.X
        dec bx
        add edi, ebx ;edi指向車子左邊的方格
    .IF BYTE PTR [edi]=="Y" ;"Y"代表空的方格，車子可以移動
        mov BYTE PTR [edi], dl
        movzx ebx, (Car PTR [esi]).len
        add edi, ebx
        mov BYTE PTR [edi], "Y"
        mov bx, (Car PTR [esi]).tail.X
        dec bx
        mov (Car PTR [esi]).tail.X, bx
        mov bx, (Car PTR [esi]).head.X
        dec bx
    .ENDIF

```

```

        mov (Car PTR [esi]).head.X, bx
    .ENDIF
.ENDIF
    .IF key==4D00h ;向右移動
        mov bx, (Car PTR [esi]).head.Y
        imul bx, 8
        add bx, (Car PTR [esi]).head.X
        inc bx
        add edi, ebx ;edi為車子右邊的方格
    .IF BYTE PTR [edi]=="Y" ;"Y"是空格，且不為出口
        mov BYTE PTR [edi], dl
        movzx ebx, (Car PTR [esi]).len
        sub edi, ebx
        mov BYTE PTR [edi], "Y"
        mov bx, (Car PTR [esi]).tail.X
        inc bx
        mov (Car PTR [esi]).tail.X, bx
        mov bx, (Car PTR [esi]).head.X
        inc bx
        mov (Car PTR [esi]).head.X, bx
    .ELSEIF BYTE PTR [edi]=="Z" ;"Z"是出口，所以移動後要把eax設為1
        mov BYTE PTR [edi], dl
        movzx ebx, (Car PTR [esi]).len
        sub edi, ebx
        mov BYTE PTR [edi], "Y"
        mov bx, (Car PTR [esi]).tail.X
        inc bx
        mov (Car PTR [esi]).tail.X, bx
        mov bx, (Car PTR [esi]).head.X
        inc bx
        mov (Car PTR [esi]).head.X, bx
        mov eax, 1
    .ENDIF
.ENDIF
    jmp FINISH_MOVE ;return

```

垂直方向移動方法雷同，但不用判斷是否移動到出口。

(3) 函式 chooseCar 判斷鍵盤輸入為 A, B, C, …，舉例來說，我選了 C 車，所以要把 C 的 target 設為 1，其他車輛的 target 設為 0，程式碼如下：

```

mov ecx, arrSize
.IF ax==1E61h ;A
    mov ebx, 0
.ENDIF

```

得到所需的陣列索引，接著：

```

L1_chooseCar:
    cmp edx, ebx
    je Equal
    mov (Car PTR [esi]).target, 0 ;非所選車輛
Next:
    add esi, TYPE Car
    inc edx
    loop L1_chooseCar
    jmp FINISH_CHOOSE
Equal:
    mov (Car PTR [esi]).target, 1 ;選擇的車輛，target為1
    jmp Next
FINISH_CHOOSE:
    ret

```

(4) 函式 drawGraph，目的是準備要輸出到螢幕的資訊，因為遊戲內部邏輯判斷使用的地圖陣列是 8*8 大小，但要輸出到螢幕的方格比較多，所以對地圖陣列裡面 64 個方格逐一對照他們的方格圖案到螢幕上。而繪製方格的方法，採用對每一個列完成，所以用到雙層迴圈，外部是高的大小，內部是寬的大小。還要把迴圈中需要的暫存器數值用堆疊儲存起來，方便迴圈正常執行。如下：

```

drawGraph PROC USES eax ebx ecx edx esi edi,
    graph: PTR BYTE,
    arr: PTR Car,
    arrSize: DWORD

    mov eax, graph
    mov esi, OFFSET text_Graph ;螢幕文字
    mov edi, OFFSET attributes_Graph ;螢幕文字色彩
    mov ecx, 64 ;地圖8*8 = 64
    mov ebx, 0 ;輸出8個方格後應換行，所以用ebx計算已經輸出的方格數目
    ;L1_drawgraph繪製每一個方格，執行8*8 = 64次
    L1_drawgraph: ;這個地方原本想用loop指令執行迴圈，但我們用jmp和手動減去ecx取代
        dec ecx
        push ebx

```

```

    .IF BYTE PTR [eax]=="N" ;地圖周圍牆壁
        mov bl, 3Dh ;文字
        mov dx, 088h ;顏色(灰)
    .ELSEIF BYTE PTR [eax]=="Y" ;可以走的空格
        mov bl, 20h
        mov dx, 077h ;顏色(白)
    .ELSEIF BYTE PTR [eax]=="Z"
        mov bl, 20h
        mov dx, 077h
    .ELSE
        mov bl, BYTE PTR [eax] ;bl是文字內容，這裡是汽車編號ABC...
        push edi
        mov edi, arr
        push ecx
        mov ecx, arrSize ;汽車陣列長度
    L2_drawgraph: ;為了找出目前車輛顏色，先搜尋汽車陣列(L2_drawgraph)
        mov bh, (Car PTR [edi]).what
        cmp bh, bl
        je Equal
        jmp Next
    Equal:
        jmp Finish_choose_drawgraph
    Next:
        add edi, TYPE Car
        loop L2_drawgraph
    Finish_choose_drawgraph:
        mov bl, (Car PTR [edi]).what ;汽車編號
        mov dx, (Car PTR [edi]).color ;汽車顏色
        pop ecx
        pop edi
    .ENDIF
    push ecx ;保留L1_drawgraph的ecx
    mov ecx, BLOCK_HEIGHT
    L_height_drawgraph: ;方格的高，迴圈
        push ecx ;保留L_height_drawgraph的ecx
        mov ecx, BLOCK_WIDTH
    L_width_drawgraph: ;方格的寬，迴圈(雙層迴圈)
        mov BYTE PTR [esi], bl

```

```

        mov bl, 20h ;每一個方格在顯示一次汽車編號後，不再顯示
        mov WORD PTR [edi], dx
        add esi, 1
        add edi, 2
        loop L_width_drawgraph
    sub esi, BLOCK_WIDTH
    add esi, BLOCK_WIDTH * 8
    sub edi, BLOCK_WIDTH * 2
    add edi, BLOCK_WIDTH * 16 ;esi和edi指向方格内部的下一列
    pop ecx ;取出L_height_drawgraph的ecx
    loop L_height_drawgraph
sub esi, MAP_WIDTH * BLOCK_HEIGHT
add esi, BLOCK_WIDTH ;esi指向下一個方格的起始位置
sub edi, MAP_WIDTH * BLOCK_HEIGHT * 2
add edi, BLOCK_WIDTH * 2 ;edi指向下一個方格的起始位置
pop ecx ;L1_drawgraph的ecx
inc eax ;地圖的下一個方格
pop ebx ;取回已經輸出的方格數目
inc ebx
cmp ebx, 8
je NextLine
Done_nextline:
    cmp ecx, 0
    jne L1_drawgraph ;不使用loop L1_drawgraph，因為程式碼間隔太長，會產生錯誤
    jmp Finish_drawgraph
NextLine:
    add esi, MAP_WIDTH * (BLOCK_HEIGHT-1)
    add edi, MAP_WIDTH * (BLOCK_HEIGHT-1) * 2
    mov ebx, 0
    jmp Done_nextline
Finish_drawgraph:
    ret
drawGraph ENDP

```

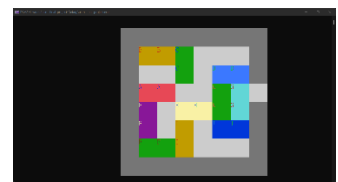
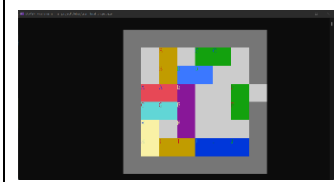
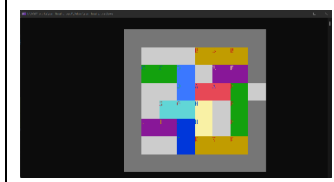
(5) 函式 screenGraph，真正輸出到螢幕，使用 WriteConsoleOutputAttribute 以及 WriteConsoleOutputCharacter，全部輸出到螢幕上。

```
screenGraph PROC USES ecx esi edi
```

```
    mov ecx, MAP_HEIGHT
```

```
mov esi, OFFSET attributes_Graph
mov edi, OFFSET text_Graph
L1_screengraph: ;每次迴圈對每一行輸出
    push ecx
    INVOKE WriteConsoleOutputAttribute,
        consoleHandle,
        esi,
        MAP_WIDTH,
        xyPosition,
        ADDR cellsWritten
    INVOKE WriteConsoleOutputCharacter,
        consoleHandle,
        edi,
        MAP_WIDTH,
        xyPosition,
        ADDR bytesWritten
    inc xyPosition.Y
    add esi, MAP_WIDTH * 2
    add edi, MAP_WIDTH
    pop ecx
    loop L1_screengraph
    sub xyPosition.Y, MAP_HEIGHT ;把xyPosition(螢幕起始位置)還原為原來的值
    ret
screenGraph ENDP
```

(C) 程式截圖

Level_1	Level_2	Level_3
		

玩家獲勝時，跳出一個訊息塊，按下確定結束程式。

