組合語言與系統程式

期末專題書面報告

專題題目：Rapid Roll

第38組組員：

資工二B 曾尚群 108502562

資工二B 林彥承 108502561

資工二B 崔家銘 108502563

資工二B 黃彥綸 108502559

分工：

Menu --------- 曾尚群

GameLoop --- 林彥承、崔家銘

報告 ---------- 黃彥綸

使用函式庫：

Irvine32

遊戲規則：

操作：使用A、D操作方塊左右移動。

遊戲過程：會不斷有平台從下往上升起，當方塊因被平台抬升或操作而超出遊戲邊界時，遊戲結束。

程式架構：

主要為Menu、GameLoop。

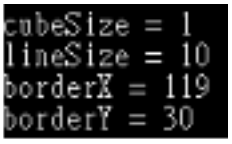
Menu包含開始畫面及結束畫面，開始畫面顯示標題、操作說明，使用任意按鍵進入遊戲或者按ESC離開程式；結束畫面顯示分數(get_score取得)使用任意按鍵回到開始畫面並使用Initialize初始化必要的變數。

GameLoop為遊戲進行：

1. drawCube玩家操縱的方塊：清除移動前位置，畫出目前位置。
2. cubeMoveLeft、cubeMoveRight、cubeMoveUp、cubeMoveDown移動方塊：記錄移動前位置，根據輸入及isGrounded決定並記錄目前位置。
3. drawLine平台：清除上升前位置，畫出目前位置。
4. lineMoveUp上升平台：記錄移動前後的位置
5. isOutOfBorder判斷方塊有無超出邊界
6. isGrounded判斷方塊是否在平台上
7. getRamdomNumber取得隨機數以生成平台

程式碼：

常數：



.data：

Menu：titleMSG為程式標題，gameMSG0~4為開始畫面標題，gameMSGPosition為開始畫面標題起始座標。

instructionMSG為開始畫面的操作說明，instructionMSGPosition為操作說明座標。

startMSG為開始畫面的開始提示，startMSGPosition為開始提示座標，startMSGAttribute記錄開始提示的顏色。

exitMSG為開始畫面的離開提示，exitMSGPosition為離開提示座標，exitMSGAttribute記錄離開提示的顏色。

```
.data
;=============================================================================
;                                For Menu /Start
;=============================================================================
titleMSG BYTE "Rapid Roll", 0
gameMSG0 BYTE "*****    *      ******   *****  *****      *****     *****   *     *    *", 0
gameMSG1 BYTE "*    *     * *    *      *    *   *    *     *    *   *     *  *     *    *", 0
gameMSG2 BYTE "*****     ******   *****      *    *    *     *****   *       * *     *    *", 0
gameMSG3 BYTE "*    *    *    *       *  *         *    *     *    *   *     *  *     *    *", 0
gameMSG4 BYTE "*     * *       * *        *****  *****      *     *   *****   ***** *****", 0
gameMSGPosition COORD <20, 5>

instructionMSG BYTE "Press A & D to move", 0
instructionMSGPosition COORD <20, 15>

startMSG BYTE "Press any key to start", 0
startMSGPosition COORD <20, 20>
startMSGAttribute WORD LENGTHOF startMSG DUP(1001b)

exitMSG BYTE "Press ESC to exit", 0
exitMSGPosition COORD <20, 22>
exitMSGAttribute WORD LENGTHOF exitMSG DUP(1100b)

;"GAME OVER"
gameOverMSG0 BYTE "  ****       *     *       * ******       *****   *       * ****** *****  ", 0
gameOverMSG1 BYTE " *          * *    **     ** *            *     *   *     *  *      *    * ", 0
gameOverMSG2 BYTE "*   ****   *****   * *   * * ******      *       *   *   *   ****** *****  ", 0
gameOverMSG3 BYTE " *    **  *     *  *  * *  * *            *     *     * *    *      *    * ", 0
gameOverMSG4 BYTE "  **** * *      * *   *   * ******       *****       *     ****** *     *", 0
gameOverMSGPosition COORD <20, 5>
gameOverMSGAttribute WORD LENGTHOF gameOverMSG0 DUP(0100b)

returnMSG BYTE "Press any key to go back to the menu", 0
returnMSGPosition COORD <20, 20>
returnMSGAttribute WORD LENGTHOF returnMSG DUP(1001b)

scoreMSG BYTE "Score: ", 0
scoreMSGPosition COORD <20, 15>
scorePosition COORD <27 ,15>

outputHandle DWORD 0
bytesWritten DWORD 0
;=============================================================================
;                                For Menu /End
;=============================================================================

;=============================================================================
;                             For Game Loop / Start
;=============================================================================
; Cube
cubeCurrentPosition COORD <50, 5>
cubePreviousPosition COORD <?, 5>
cubeBody BYTE ' '
; Line
line1 COORD <50, 15>
line1Previous COORD <>
line2 COORD <40,10>
line2Previous COORD <>
line3 COORD <60,20>
line3Previous COORD <>
line4 COORD <70,30>
line4Previous COORD <>
line5 COORD <75,40>
line5Previous COORD <>
lineBody BYTE lineSize DUP('_')
removeLineBody BYTE lineSize DUP(' ')
lineLength DWORD lineSize
;Random Number
randomNum DWORD ?
; Score
score DWORD 0
startTime DWORD ?
; Game Tik
TICK DWORD 0
; Border
border COORD <100, 30>
isOut DWORD 0
; Ground
isOnTheGround DWORD 0
whichLine DWORD ?
;Draw
cellsWritten BYTE ?
;=============================================================================
;                             For Game Loop /End
;=============================================================================
```

gameOverMSG0~4為結束畫面標題,
gameOverMSGPosition為結束畫面標題起始座標,
gameOverMSGAttribute記錄結束畫面標題的顏色。

returnMSG為結束畫面的回到開始畫面提示,
returnMSGPosition為提示位置，returnMSGAttribute記錄提示的顏色。

scoreMSG為分數訊息，scoreMSGPosition為分數訊息位置，scorePosition為分數位置。

outputHandle用來存STD_OUTPUT_HANDLE,
bytesWritten存WriteConsoleOutputCharacter回傳值。

Game Loop：cubeCurrentPosition存玩家目前位置,
cubePreviousPosition存玩家上個位置，cubeBody為玩家方塊" "。

line1~5為平台座標，line1~5Previous為平台前座標,
lineBody為10個"_"用來做出平台，removeLineBody為10個" "用來消除平台，lineLength為10。

randomNum存getRandomNumber取得的隨機數。

score存分數，startTime存開始時間。

TICK存Game Loop的循環數。

border為邊界座標，isOut存是否超出邊界的判斷值。

isOnTheGround存方塊是否在平台上的判斷值，whichLine存方塊所在的平台編號。

cellsWritten存WriteConsoleOutputCharacter回傳值。

.code：

initialize

使用ax初始化方塊和平台1~5的位置。

使用eax=0初始化分數、超出邊界判斷值、Game Loop循環數和玩家在平台上的判斷值。

```
;=============================================================;
;                  Initialize the variables                   ;
;=============================================================;
initialize PROC USES eax
    mov ax, 50
    mov cubeCurrentPosition.X, ax
    mov line1.X, ax
    mov ax, 5
    mov cubeCurrentPosition.Y, ax
    mov cubePreviousPosition.Y, ax
    mov ax, 15
    mov line1.Y, ax
    mov ax, 40
    mov line2.X, ax
    mov ax, 10
    mov line2.Y, ax
    mov ax, 60
    mov line3.X, ax
    mov ax, 20
    mov line3.Y, ax
    mov ax, 70
    mov line4.X, ax
    mov ax, 30
    mov line4.Y, ax
    mov ax, 75
    mov line5.X, ax
    mov ax, 40
    mov line5.Y, ax

    xor eax, eax
    mov score, eax
    mov isOut, eax
    mov TICK, eax
    mov isOnTheGround, eax
    ret
initialize ENDP
```

```
;=============================================
;                   Move Cube
;=============================================
; Left
cubeMoveLeft PROC USES ax bx
    .IF cubeCurrentPosition.X > 1
        mov ax, cubeCurrentPosition.X
        mov bx, cubeCurrentPosition.Y
        mov cubePreviousPosition.X, ax
        mov cubePreviousPosition.Y, bx
        sub cubeCurrentPosition.X, 1
    .ENDIF
    ret
cubeMoveLeft ENDP

; Right
cubeMoveRight PROC USES ax bx
    .IF cubeCurrentPosition.X < borderX
        mov ax, cubeCurrentPosition.X
        mov bx, cubeCurrentPosition.Y
        mov cubePreviousPosition.X, ax
        mov cubePreviousPosition.Y, bx
        add cubeCurrentPosition.X, 1
    .ENDIF
    ret
cubeMoveRight ENDP

; Up or Down
cubeMoveUp PROC USES ax bx

    mov ax, cubeCurrentPosition.X
    mov bx, cubeCurrentPosition.Y
    mov cubePreviousPosition.X, ax
    mov cubePreviousPosition.Y, bx
    .IF isOnTheGround == 1
        .IF whichLine == 1
            mov ax, line1.Y
            mov cubeCurrentPosition.Y, ax
        .ELSEIF whichLine == 2
            mov ax, line2.Y
            mov cubeCurrentPosition.Y, ax
        .ELSEIF whichLine == 3
            mov ax, line3.Y
            mov cubeCurrentPosition.Y, ax
        .ELSEIF whichLine == 4
            mov ax, line4.Y
            mov cubeCurrentPosition.Y, ax
        .ELSEIF whichLine == 5
            mov ax, line5.Y
            mov cubeCurrentPosition.Y, ax
        .ENDIF
    .ELSE
        sub cubeCurrentPosition.Y, 1
    .ENDIF

    ret
cubeMoveUp ENDP

cubeMoveDown PROC USES ax bx

    mov ax, cubeCurrentPosition.X
    mov bx, cubeCurrentPosition.Y
    mov cubePreviousPosition.X, ax
    mov cubePreviousPosition.Y, bx
    add cubeCurrentPosition.Y, 1

    ret
cubeMoveDown ENDP
```

cubeMoveLeft

如果玩家目前座標未超出左邊界，則把玩家目前座標透過ax、bx存到玩家前座標，玩家目前X座標-1(玩家座標左移)。

cubeMoveRight

如果玩家目前座標未超出右邊界，則把玩家目前座標透過ax、bx存到玩家前座標，玩家目前X座標+1(玩家座標右移)。

cubeMoveUp

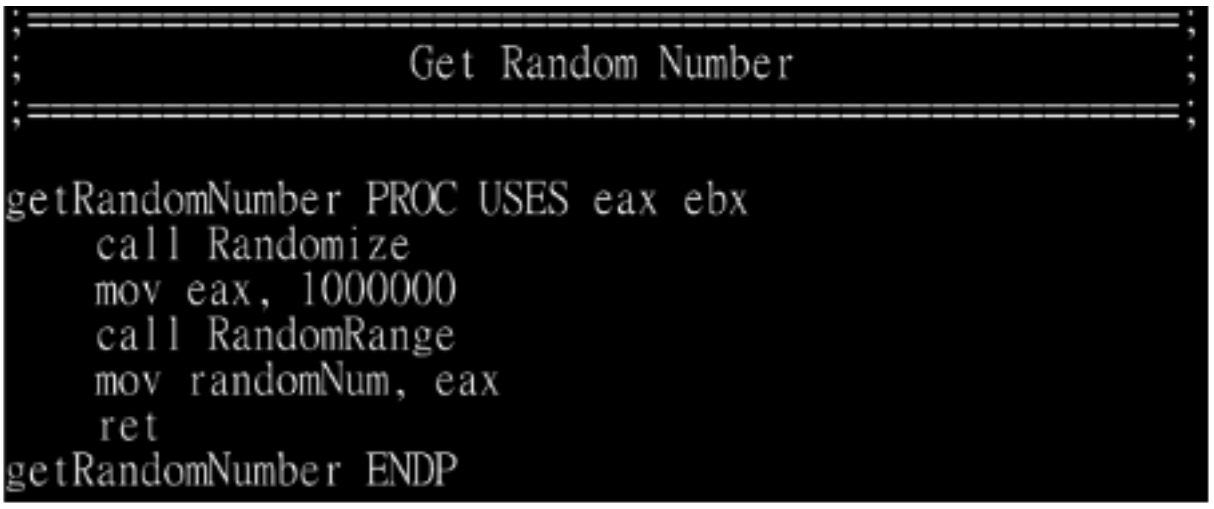
把玩家目前座標透過ax、bx存到玩家前座標，如果玩家在平台上，則把玩家的Y座標設為所在平台的Y座標；其他則玩家Y座標-1(上移)。

cubeMoveDown

把玩家目前座標透過ax、bx存到玩家前座標，玩家Y座標+1(下移)。

getRandomNumber

取得0~999999的隨機數並存到變數randomNum中。



```
;=====================================================
;                  Get Random Number                  ;
;=====================================================

getRandomNumber PROC USES eax ebx
    call Randomize
    mov eax, 1000000
    call RandomRange
    mov randomNum, eax
    ret
getRandomNumber ENDP
```

lineMoveUp

分別控制line1~5上升。

把平台目前座標透過ax、bx存到平台前座標，如果還未到達上邊界，則平台目前Y座標-1(上移)；

到達上邊界時，則用getRandomNumber取得隨機數存進eax，讓ax/60，dx最後相當於餘數*2，再存進平台目前X座標，平台目前Y座標設為最底層Y。

```
;=============================================;
;                   Move Line                 ;
;=============================================;

lineMoveUp PROC USES eax ebx ecx edx
    ; move line 1
    mov ax, line1.X
    mov bx, line1.Y
    mov line1Previous.X, ax
    mov line1Previous.Y, bx
    .IF line1.Y > 0
        dec line1.Y
    .ELSE
        invoke getRandomNumber
        xor edx, edx
        mov eax, randomNum
        mov ax, ax
        mov bx, 60
        div bx
        shl dx, 1
        mov line1.X, dx
        mov line1.Y, 30
    .ENDIF
    ; move line 2
    mov ax, line2.X
    mov bx, line2.Y
    mov line2Previous.X, ax
    mov line2Previous.Y, bx
    .IF line2.Y > 0
        dec line2.Y
    .ELSE
        invoke getRandomNumber
        xor edx, edx
        mov eax, randomNum
        mov ax, ax
        mov bx, 60
        div bx
        shl dx, 1
        mov line2.X, dx
        mov line2.Y, 30
    .ENDIF
    ; move line 3
    mov ax, line3.X
    mov bx, line3.Y
    mov line3Previous.X, ax
    mov line3Previous.Y, bx
    .IF line3.Y > 0
        dec line3.Y
    .ELSE
        invoke getRandomNumber
        xor edx, edx
        mov eax, randomNum
        mov ax, ax
        mov bx, 60
        div bx
        shl dx, 1
        mov line3.X, dx
        mov line3.Y, 30
    .ENDIF
    ; move line 4
    mov ax, line4.X
    mov bx, line4.Y
    mov line4Previous.X, ax
    mov line4Previous.Y, bx
    .IF line4.Y > 0
        dec line4.Y
    .ELSE
        invoke getRandomNumber
        xor edx, edx
        mov eax, randomNum
        mov ax, ax
        mov bx, 60
        div bx
        shl dx, 1
        mov line4.X, dx
        mov line4.Y, 30
    .ENDIF
    ; mov line 5
    mov ax, line5.X
    mov bx, line5.Y
    mov line5Previous.X, ax
    mov line5Previous.Y, bx
    .IF line5.Y > 0
        dec line5.Y
    .ELSE
        invoke getRandomNumber
        xor edx, edx
        mov eax, randomNum
        mov ax, ax
        mov bx, 60
        div bx
        shl dx, 1
        mov line5.X, dx
        mov line5.Y, 30
    .ENDIF
    ret
lineMoveUp ENDP
```

drawCube

設定游標位置為方塊前座標，輸出黑底空格覆蓋原方塊，eax存進stack。

設定游標位置為方塊目前座標，輸出淡青綠空格做出方塊，pop出eax設定文字顏色和底色為預設。

drawLine

使用WriteConsoleOutputCharacter輸出空格清除原line1~5，一樣用WriteConsoleOutputCharacter輸出新的line1~5

```
;-------------------------------------------------;
;                    Draw Line                    ;
;-------------------------------------------------;

drawLine PROC
    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR removeLineBody,
      lineLength,
      line1Previous,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR removeLineBody,
      lineLength,
      line2Previous,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR removeLineBody,
      lineLength,
      line3Previous,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR removeLineBody,
      lineLength,
      line4Previous,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR removeLineBody,
      lineLength,
      line5Previous,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR lineBody,
      lineLength,
      line1,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR lineBody,
      lineLength,
      line2,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR lineBody,
      lineLength,
      line3,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR lineBody,
      lineLength,
      line4,
      ADDR cellsWritten

    invoke WriteConsoleOutputCharacter,
      outputHandle,
      ADDR lineBody,
      lineLength,
      line5,
      ADDR cellsWritten
    ret
drawLine ENDP
```

isOutOfBorder

當玩家目前X座標在左右邊界上時，變數isOut設為1；玩家目前Y座標超出上下邊界時，變數isOut設為1；上述都為否，則變數isOut設為0。

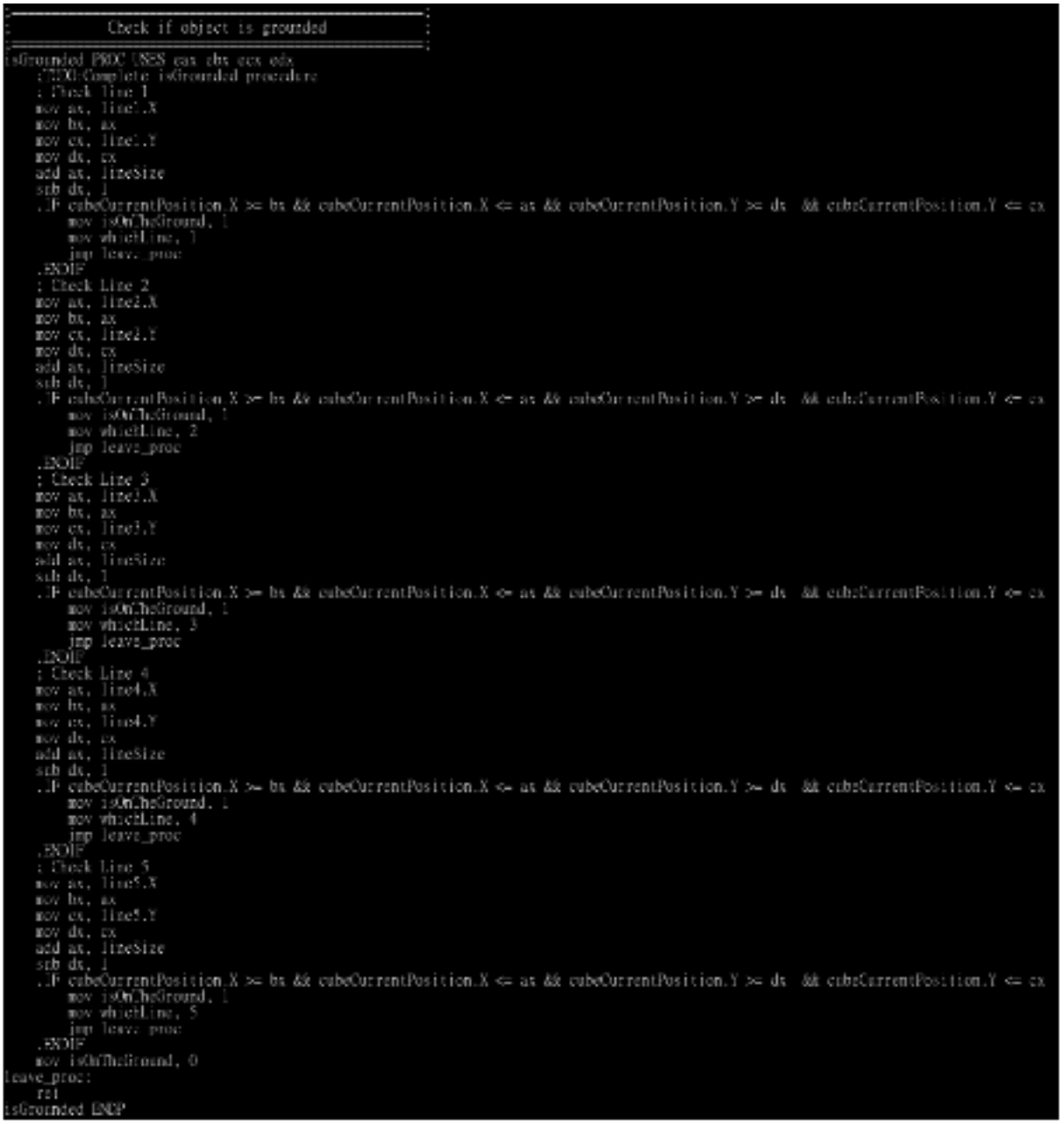
```
;=====================================================;
;          Check if object is out of border           ;
;=====================================================;
isOutOfBorder PROC
    .IF cubeCurrentPosition.X >= borderX
        mov isOut, 1
    .ELSEIF cubeCurrentPosition.X <= 1
        mov isOut, 1
    .ELSEIF cubeCurrentPosition.Y > borderY
        mov isOut, 1
    .ELSEIF cubeCurrentPosition.Y < 1
        mov isOut, 1
    .ELSE
        mov isOut, 0
    .ENDIF
    ret
isOutOfBorder ENDP
```

isGrounded

檢查方塊是否在平台上並判斷是哪一平台。

從line1依序到line5檢查，使用ax、bx、cx、dx記錄平台本身及上方一格的矩形空間座標，當玩家目前座標位於矩形空間內，則變數isOnTheGround設為1，變數whichLine設為所在平台編號。

當方塊不在平台上時，isOnTheGround設為0。



```
;-----------------------------------------;
;          Check if object is grounded    ;
;-----------------------------------------;
isGrounded PROC USES eax ebx ecx edx
    ;TODO:Complete isGrounded procedure
    ; Check line 1
    mov ax, line1.X
    mov bx, ax
    mov cx, line1.Y
    mov dx, cx
    add ax, lineSize
    sub dx, 1
    .IF cubeCurrentPosition.X >= bx && cubeCurrentPosition.X <= ax && cubeCurrentPosition.Y >= dx  && cubeCurrentPosition.Y <= cx
        mov isOnTheGround, 1
        mov whichLine, 1
        jmp leave_proc
    .ENDIF
    ; Check Line 2
    mov ax, line2.X
    mov bx, ax
    mov cx, line2.Y
    mov dx, cx
    add ax, lineSize
    sub dx, 1
    .IF cubeCurrentPosition.X >= bx && cubeCurrentPosition.X <= ax && cubeCurrentPosition.Y >= dx  && cubeCurrentPosition.Y <= cx
        mov isOnTheGround, 1
        mov whichLine, 2
        jmp leave_proc
    .ENDIF
    ; Check Line 3
    mov ax, line3.X
    mov bx, ax
    mov cx, line3.Y
    mov dx, cx
    add ax, lineSize
    sub dx, 1
    .IF cubeCurrentPosition.X >= bx && cubeCurrentPosition.X <= ax && cubeCurrentPosition.Y >= dx  && cubeCurrentPosition.Y <= cx
        mov isOnTheGround, 1
        mov whichLine, 3
        jmp leave_proc
    .ENDIF
    ; Check Line 4
    mov ax, line4.X
    mov bx, ax
    mov cx, line4.Y
    mov dx, cx
    add ax, lineSize
    sub dx, 1
    .IF cubeCurrentPosition.X >= bx && cubeCurrentPosition.X <= ax && cubeCurrentPosition.Y >= dx  && cubeCurrentPosition.Y <= cx
        mov isOnTheGround, 1
        mov whichLine, 4
        jmp leave_proc
    .ENDIF
    ; Check Line 5
    mov ax, line5.X
    mov bx, ax
    mov cx, line5.Y
    mov dx, cx
    add ax, lineSize
    sub dx, 1
    .IF cubeCurrentPosition.X >= bx && cubeCurrentPosition.X <= ax && cubeCurrentPosition.Y >= dx  && cubeCurrentPosition.Y <= cx
        mov isOnTheGround, 1
        mov whichLine, 5
        jmp leave_proc
    .ENDIF
    mov isOnTheGround, 0
leave_proc:
    ret
isGrounded ENDP
```

get_score

用GetMsecond取得現在時間，減去變數startTime中存的時間值，再存到變數score中。

```
;=============================================================;
;                           Score                             ;
;=============================================================;
get_score PROC USES eax
    call GetMseconds
    sub eax, startTime
    mov score, eax
    ret
get_score ENDP
```

GameLoop

使用Clrscr清除畫面，開始WHILE迴圈。

WHILE {

判斷是否在平台上，讀取輸入a、d，作左右移動，再次判斷是否在平台上。

edx歸零，eax設為變數TICK再除以375，edx為餘數，如果edx為0(相當於每375次作一次判定)，則方塊在平台上時，方塊和平台都往上；方塊不在平台上時，方塊往下，平台往上。TICK設為0。

更新方塊和平台，方塊超出邊界時結束GameLoop；否則TICK+1。 }

```
;=============================================================;
;                          GameLoop                           ;
;=============================================================;
GameLoop PROC USES eax ebx ecx edx
    call Clrscr
    .WHILE TRUE
        invoke isGrounded
        call ReadKey
        .IF ax == 1E61h
            ;TODO: move cube left
            invoke cubeMoveLeft
            invoke isGrounded
        .ELSEIF ax == 2064h
            ;TODO: move cube right
            invoke cubeMoveRight
            invoke isGrounded
        .ENDIF
        ; Move Line & Cube Up
        xor edx, edx
        mov eax, TICK
        mov ebx, 375
        div ebx
        .IF edx == 0
            .IF isOnTheGround == 1
                invoke lineMoveUp
                invoke cubeMoveUp
            .ELSEIF isOnTheGround == 0
                 invoke lineMoveUp
                 invoke cubeMoveDown
            .ENDIF
           mov TICK, 0
        .ENDIF
        invoke drawLine
        invoke drawCube
        invoke isOutOfBorder
        .IF isOut == 1
            jmp leave_proc
        .ENDIF
        inc TICK
    .ENDW
    leave_proc:
    ret
GameLoop ENDP
```

main

```
;=====================================================
;                       Main
;=====================================================
ain PROC
ewGame:
    INVOKE GetStdHandle, STD_OUTPUT_HANDLE
    mov outputHandle, eax
    CALL Clrscr

    INVOKE SetConsoleTitle, ADDR titleMSG

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameMSG0,
        LENGTHOF gameMSG0,
        gameMSGPosition,
        ADDR bytesWritten

    inc gameMSGPosition.Y
    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameMSG1,
        LENGTHOF gameMSG1,
        gameMSGPosition,
        ADDR bytesWritten

    inc gameMSGPosition.Y
    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameMSG2,
        LENGTHOF gameMSG2,
        gameMSGPosition,
        ADDR bytesWritten

    inc gameMSGPosition.Y
    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameMSG3,
        LENGTHOF gameMSG3,
        gameMSGPosition,
        ADDR bytesWritten

    inc gameMSGPosition.Y
    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameMSG4,
        LENGTHOF gameMSG4,
        gameMSGPosition,
        ADDR bytesWritten
    sub gameMSGPosition.Y, 4

    INVOKE  WriteConsoleOutputCharacter,
        outputHandle,
        ADDR instructionMSG,
        LENGTHOF instructionMSG,
        instructionMSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR startMSGAttribute,
        LENGTHOF startMSG,
        startMSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR startMSG,
        LENGTHOF startMSG,
        startMSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR exitMSGAttribute,
        LENGTHOF exitMSG,
        exitMSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR exitMSG,
        LENGTHOF exitMSG,
        exitMSGPosition,
        ADDR bytesWritten

    xor ax, ax
aitForInput:
    CALL ReadChar
    CMP ax, 0       ;There is no key pressed
    JE waitForInput
    CMP ax, 011Bh   ;The user presses ESC
    JE exitGame
```

(label)newGame:

取得STD_OUTPUT_HANDLE存進outputHandle中，Clrscr清除畫面，改變程式標題，顯示開始畫面標題、操作說明、開始提示、離開提示。

ax歸零，等待玩家輸入任意按鍵進入遊戲，輸入ESC則離開程式。

(label)start:

eax存進stack，取得開始時間並存進變數startTime，eax設為原本的值。

GameLoop開始。

```
start:
;=====================================================
;                start the game loop                  ;
;=====================================================
    push eax
    call GetMseconds ; get time to count score
    mov startTime, eax
    pop eax

    invoke GameLoop
```

(label)gameOver:

Clrscr清除畫面，顯示結束畫面標題、分數、回主畫面提示。

ax歸零，最後等待玩家輸入任意按鍵。

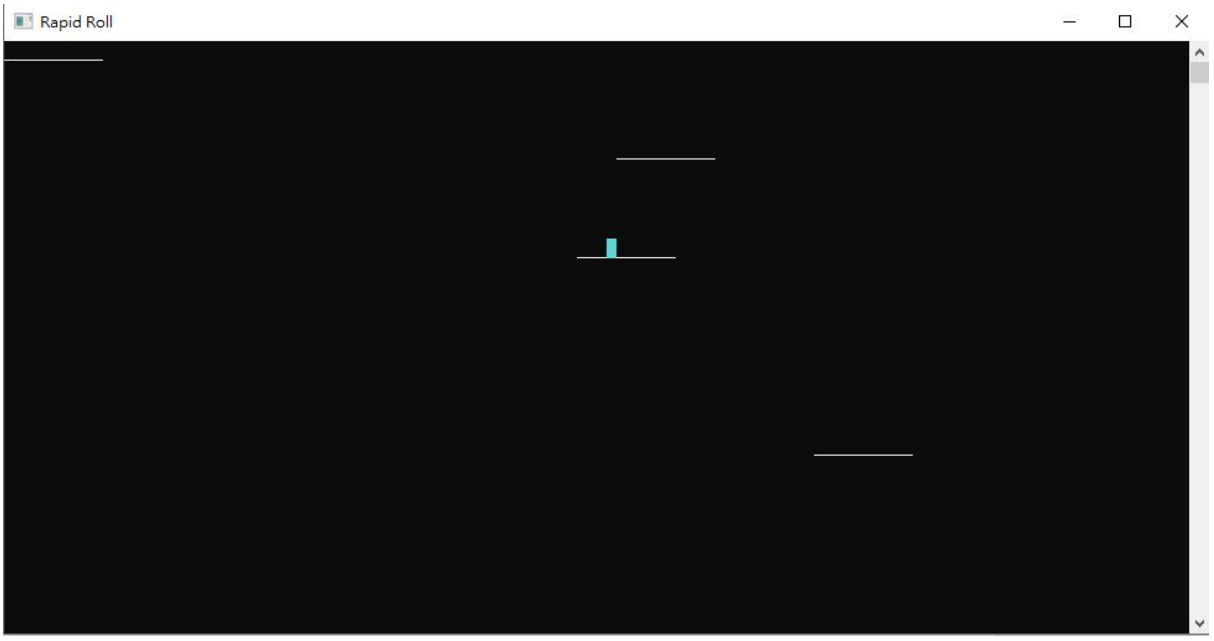
```
gameOver:
    CALL Clrscr

    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR gameOverBSGAttribute,
        LENGTHOF gameOverBSG0,
        gameOverBSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameOverBSG0,
        LENGTHOF gameOverBSG0,
        gameOverBSGPosition,
        ADDR bytesWritten

    inc gameOverBSGPosition.Y
    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR gameOverBSGAttribute,
        LENGTHOF gameOverBSG1,
        gameOverBSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameOverBSG1,
        LENGTHOF gameOverBSG1,
        gameOverBSGPosition,
        ADDR bytesWritten

    inc gameOverBSGPosition.Y
    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR gameOverBSGAttribute,
        LENGTHOF gameOverBSG2,
        gameOverBSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameOverBSG2,
        LENGTHOF gameOverBSG2,
        gameOverBSGPosition,
        ADDR bytesWritten

    inc gameOverBSGPosition.Y
    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR gameOverBSGAttribute,
        LENGTHOF gameOverBSG3,
        gameOverBSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameOverBSG3,
        LENGTHOF gameOverBSG3,
        gameOverBSGPosition,
        ADDR bytesWritten

    inc gameOverBSGPosition.Y
    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR gameOverBSGAttribute,
        LENGTHOF gameOverBSG4,
        gameOverBSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR gameOverBSG4,
        LENGTHOF gameOverBSG4,
        gameOverBSGPosition,
        ADDR bytesWritten
    sub gameOverBSGPosition.Y, 4

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR scoreBSG,
        LENGTHOF scoreBSG,
        scoreBSGPosition,
        ADDR bytesWritten

    ;print the score
    INVOKE SetConsoleCursorPosition,
        outputHandle,
        scorePosition
     call get_score
     mov eax, score
     CALL WriteDec

    INVOKE WriteConsoleOutputAttribute,
        outputHandle,
        ADDR returnBSGAttribute,
        LENGTHOF returnBSG,
        returnBSGPosition,
        ADDR bytesWritten

    INVOKE WriteConsoleOutputCharacter,
        outputHandle,
        ADDR returnBSG,
        LENGTHOF returnBSG,
        returnBSGPosition,
        ADDR bytesWritten

    xor ax, ax
waitTillInput:
    CALL ReadChar
    CMP ax, 0
    JE waitTillInput
```

按下任意鍵後，會使用initialize初始化變數並回到一開始的newGame。

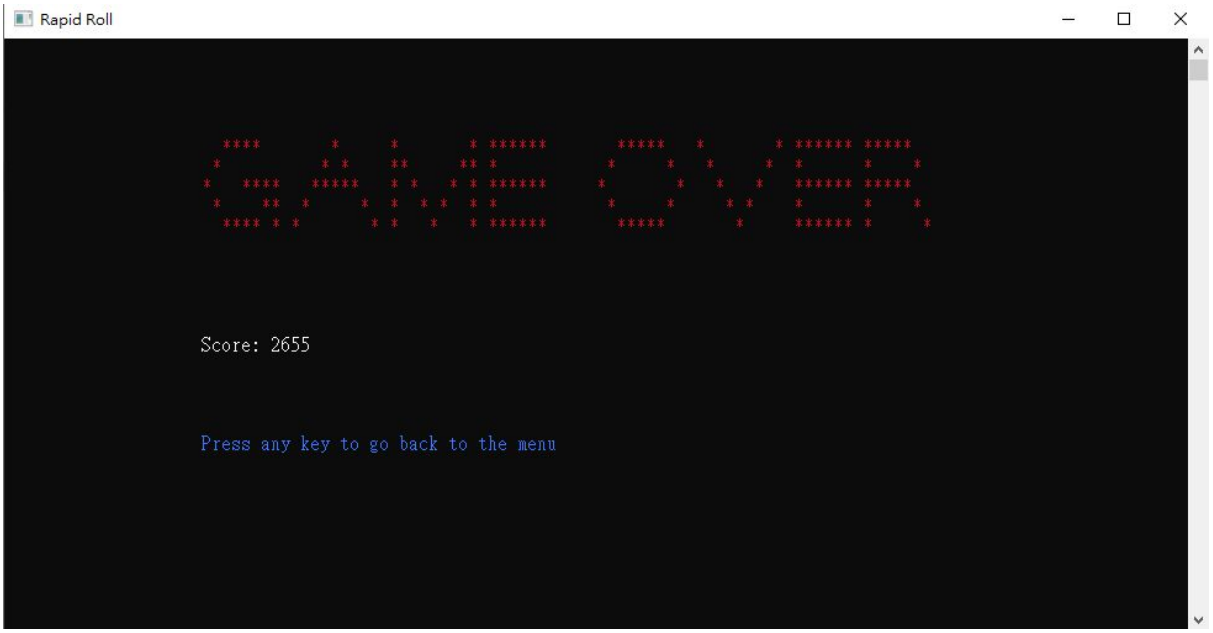
```
;=============================================================;
;                    Reset the game                           ;
;          Reset the old variables if necessary               ;
;=============================================================;
    CALL initialize
    JMP newGame

exitGame:
main ENDP
END main
```

遊戲開始畫面：

遊戲進行畫面：



遊戲結束畫面：



分數為存活時間(ms)

更新：

修正Demo時的閃退bug