Quyo Quyo

B95902108 馮俊菘
B95901207 陳柏龍
B95901183 尤孝庭

感謝名單

J Vijn.
the author of Tonc reference
http://www.coranac.com/tonc/text/toc.htm contains

Joseph Wen
cyy
小金助教
蕭老
牡蠣
coo1be
Favonia
scan
mt
ubiquitin

設計理念

高中時曾經玩過這款遊戲，最近又看到有人在玩，就想在 GBA 上寫一個。但是因為 GBA 上按鍵不足，機器大小也不適合讓兩人在同一台 GBA 上對戰，因此我們決定寫「單人連鎖磨練版」。

遊戲方法

遊戲開始的選單可以選擇要幾種顏色的泡泡（3~5 種）。進入遊戲以後，會出現一個 12 列 6 欄的方框，初始每格都是空的。上面會不斷放出泡泡（在第三欄），每次兩個。方框旁邊會有四格是待會要放出的泡泡。使用者可以按左右鍵移動、按 AB 鍵旋轉目前正在落下的泡泡，或是按 START 暫停。

剛放出的泡泡會等速落下，當泡泡落到穩定位置（碰到方框下緣或是碰到其他泡泡）就會停止。此時遊戲會檢查是否有四個以上相同顏色的泡泡擺在相鄰的位置，若有則把這些泡泡消除，並讓這些泡泡上方的其他泡泡落下，然後再次進行這個檢查（連鎖消除），直到沒有泡泡可以消除時為止。才會再從第三欄放出下兩個泡泡。

遊戲會記錄當前的分數和最大連鎖數，如果最大連鎖數超過這一關的目標，則表示過關。使用者會進入下一關（更高的連鎖數目標），如此持續下去。如果第三欄已被放滿，放出的泡泡無處容身，則遊戲結束，使用者回到主選單。

實作內容

我們參考 Tonc library 網站上的介紹，特別是針對 control registers 的那些，做出我們需要的 library。在整個 project 中，只有取亂數時（srand()、rand()）有用到 stdlib.h，其他都是自己定義的 constant 和 library。

在實作過程中，我們用 bubble array（在 object.cpp）來記錄現在出現的泡泡，當泡泡被消掉時清除。因為泡泡會一直出現，為了避免”array index out of bound”，我們在檢查泡泡是否被消掉時，會呼叫 delete_bubble() 把這個泡泡移出 bubble array（移到 array 最後端）。

另外我們還用一個 bubble_slot array（在 control.cpp）來記錄出現在格子上的泡泡。當一個泡泡落下以後，才會記錄在 bubble_slot 裡面，而且在落下、消除等過程中，泡泡會移出 bubble_slot，直到他落到定點才會再記錄進去。除了 bubble_slot 以外，其他還有 active 和 prepare 兩個 array 分別存放正在移動中（使用者控制中）和準備區的泡泡。

顯示模式：

我們用 mode4 來顯示畫面，背景部份使用 bitmap，泡泡、數字及動畫都使用 sprite。因為 mode4 的關係，只有一半的 sprite tile memory 可以用，幸好我們需要的 sprite tile memory 也不多，所以一切安好。

前景物件：

我們的 sprite 有三種（bubble, number, menu_bubble），都屬於同一個型態 Object。（如右）

```
typedef struct
{
    u16 X , Y ;
    u16 COLOR ;
    u16 OAM_IND ;
} Object ;
```

GBA 可用的 OAM 記憶體只有 128 組，意味著只能同時顯示最多 128 個 sprite，為此，我們寫了幾個簡單的 OAM management function 來管理每個 Object 的 OAM 位址：

get_OAM_ind() 和 rel_OAM_ind() 。

我們寫了一個 draw_object() 來畫出需要被顯示的 sprite，draw_bubble()、draw_number()、draw_menu_bubble()、draw_bubble_dead()等 function 都會呼叫 draw_object()。另外還有 spr_palette_initialize()及 sprite_initialize()分別處理 sprite 的 palette 及 tile。

動畫：

在整個遊戲中我們用了很多動畫，包括 menu 上的泡泡，還有遊戲中泡泡的旋轉、消除等等。這些都是預先寫好路徑，然後用 timer 控制時間，來決定現在要把這個 object 放在哪個位置。

其中用 timer 控制時間，是用兩個 timer 搭配（TM_CASCADE），當第一個 timer 發生 overflow 的時候，第二個 timer 就會 tick 一次。這樣只需要控制第一個 timer 什麼時候會 overflow，在第二個 timer 上可以定出我們想要的時間。

聲音：

我們用 channel 1 來放聲音，在 macro 裡面定義了 sound_initialize() 和 play_sound_note() 兩個 function 來初始化 sound registers 和播放單一音符，其中音符所對應的頻率是參照 Tonc library 裡面給的常數。

程式碼：（這裡列出幾個重要的 function）

control.cpp / control.h

控制一場遊戲的流程，主要的遊戲程式都放在這邊。

attack()：

檢查有哪些泡泡會被消掉，並呼叫 attack_disappear()。

attack_disappear()：

當有泡泡消除或升級條件達成時呼叫，讓泡泡閃動，並沿著既定路徑（fly array）播放動畫與音效，最後呼叫 remove_bubble() 把泡泡刪除。

bubble_fall()：

當 gameover 時呼叫，讓所以格子上的泡泡掉出螢幕。

control()：

遊戲主程式，負責初始設定，接收使用者輸入，並適時呼叫其他 function。

drop()：

讓所有尚未落到定位的泡泡歸位，其中播放動畫，使泡泡以等加速度平滑的下降。

get_collect() / set_collect()：

用 DFS (depth first search) 檢查是否有四顆以上同色泡泡相連（get），標上 SELECTED（set）。

rotate() / active_rotate()：

處理泡泡的旋轉，決定現在正在移動的泡泡（active[2]）將要旋轉到的位置，並呼叫 active_rotate()來處理旋轉的動畫。

definitions.h

把所有的 GBA 的 control register address 及 bitwise mask 定義成簡單易懂的常數，並且在定義 address 時先做好轉型；另外還定義了遊戲聲音的音符頻率。

game.cpp

遊戲最外層的控制，也是程式開始的地方。

initialize()：

初始化背景、聲音及 sprite 的設定。

menu()：

畫出選擇遊戲泡泡種類數量的選單，並播放動畫（menu_bubble）。

start()：

呼叫 menu()並且紀錄其執行時間（使用者按下 START 的時間）作為往後需要做亂數時的種子。

show_level()：

繪出現在 level 多少的畫面並等待使用者按下 START 繼續。

show_gameover()：

繪出 gameover 的畫面並等待使用者按下 START 繼續。

main():

整個遊戲最外層的控制，把 initialize()、menu()、control()、show_level()及 show_gameover()兜起來放在一個迴圈裡。

macro.cpp / macro.h

存放控制鍵盤、音樂、影像等與 GBA 直接溝通的函式。與 difinitions.h 配合讓其它檔案不需要出現跟 GBA 直接相關的程式碼。

object.cpp / object.h

我們在這裡定義了所有和 Object 直接相關的函數。遊戲中的每一顆泡泡，都由這個檔案中的 Object bubble[128] 紀錄其所有參數。另外還有一些 draw_digit()、hide_bubble()及 delete_bubble()等函式，在此就不加贅述。

get_OAM_ind() / rel_OAM_ind():

這兩個函式前面已經提到，是用來管理 Object 的 OAM 位址。

set_object():

設定一個 Object 的起始值，包括 x,y 座標、color（也就是 tile index）、還有 OAM_ind（這個 Object 在 OAM 裡面的 index）。

draw_object():

在指定的座標畫出指定的 Object，並且設定其 priority。

general.asm

這是我們遊戲中所有 ARM assembly code 之所在。因為某些原因，這個檔案並不大，後面會詳加說明。

DRAW_BG：(在 macro.h)

傳入想要用的 palette 和想要畫的 image，畫到 VRAM 上。因為 bitmap 的填色需要 160*240 = 38400 次存取才能完成，相當費時，所以這個 function 被從 macro.cpp 裡獨立出來做加速。

SET_TILE：(在 macro.h)

只有在 sprite_initialize()時會用，把 sprite_inputs.h 裡面的所有 sprite（包括 bubble 和 number）放到 TILE RAM 裡面。因為所有的 sprite 剛好都是 16*16，所以就省略長度的參數。

DRAW_OBJECT：(在 object.h)

在顯示所有 sprite 的時候都會用到，把一個 TILE 以指定的 x,y 座標和 priority 放到指定的 OAM 位置。因為遊戲中出現了一些動畫，這個也會常常用到，所以用 assembly 來實作。

other graphic inputs

bitmap_inputs.h:

所有 bitmap 的資料都存在這裡，裡面有每張圖的調色盤（因為是使用

mode4，256 色）及每張 bitmap 的 color map。這個檔案裡的資料都是用以下程式 generate 出來：gfx2gba -fsrc xxx.bmp

sprite_inputs.h：

所有 sprite 的 tile 和調色盤都存在這裡，由以下的程式 generate 出來：gfx2gba -fsrc -t8 xxx.bmp

遭遇困難

1. 不能輸出 debug 訊息

因為 GBA 上面沒有內建顯示文字的功能，我們也沒有用任何 library，一開始就不知道如何 debug。後來找到的方法是：更改背景顏色（BMP_PRAM[0]）來表示 error 狀況，雖然沒有 debug 訊息來的明確，但也足以達成 debug 的功能。

2. 傳入五個參數給 assembly function 還有它的傳回值

DRAW_OBJECT 這個 function 需要用到五個參數，還有 SET_TILE 需要傳回 offset。這些是之前寫作業的時候沒有遇過的，我們去找了一些參考資料才把它做出來。

3. 大量使用 timer 造成 assembly 空間不大

在我們的遊戲中，很多動作、動畫都是透過 timer 控制的。因此除了幾個效率差異較大的 function（例如和「顯示畫面」相關的 function）以外，其他的有沒有轉成 assembly 都看不出什麼影響。

4. 組員間 coding style 差異

每個人的 coding style 都有些許不同，因此在寫 code 過程中，常常為了要不要空白、要不要換行等事情意見不合，造成 code 被反覆修改，浪費了不少時間。不過還好後來都得到妥協，有了共同的 style。

學到了啥？

這是我們第一次撰寫在 GBA 的平台上執行的程式。老實說，與其說是跟 ARM assembly 奮鬥，不如說是跟 GBA architecture 奮鬥。因此，這次的 project 讓我們瞭解了很多 GBA 的運作方式，也知道了不少硬體和軟體間溝通的方式。

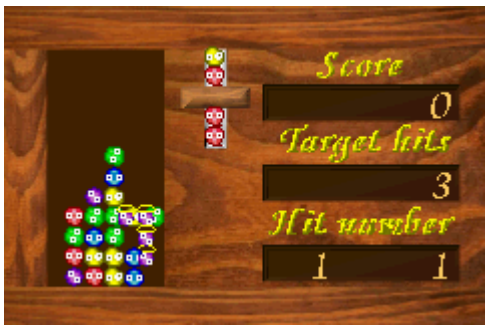
在撰寫 C++主程式要用的 assembly function 時遇到了幾個問題，在上面已經討論過。解決這些問題也花了我們一些時間，基本上算是複習了期中寫作業時的 ARM assembly，也瞭解如何使 C/C++程式和 ARM assembly 如何溝通。

從前寫程式幾乎都是自己一個人寫，很少有跟同伴合作的機會。在這次的

project 裡，為了保持溝通良好和往後的作業方便，我們在撰寫程式碼時都儘量保持著最好的 coding style。在 bug 出現時馬上停止工作，找出 bug 後再繼續。漸漸地形成了一種很有效率的合作模式，不知道在撰寫程式碼的人數增多時，這種模式還有沒有用。不過，起碼這種精神態度是身為一個 project producer 應該培養的。

以下是一些遊戲的截圖：

遊戲主選單



泡泡消除

泡泡堆疊

過關畫面