Amazeing 迷宮之旅
Report

一、遊戲介紹

這是一個兩人遊戲，雙方玩家迷失在迷宮之中，為了尋找嵌入式的密寶而一同踏上旅程。

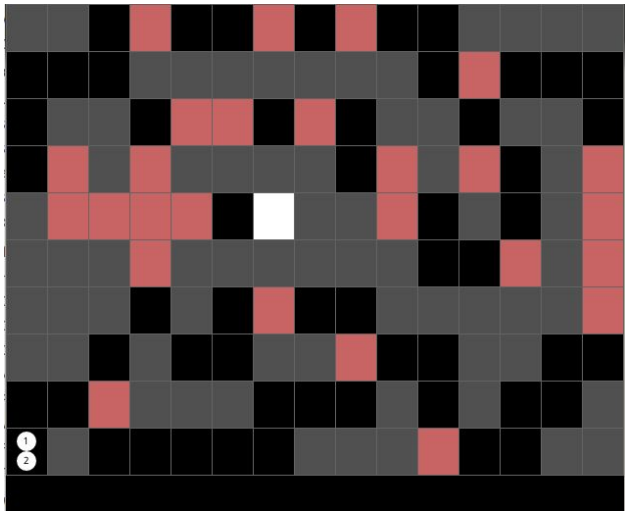
遊戲畫面顯示於PC螢幕上，遊戲開始時，兩人一起出現在迷宮的起點，雙方輪流透過鍵盤操作投擲骰子及控制前進。由於每次一定要剛好移動骰子擲出的步數，有時即使終點在前也不一定能輕鬆獲勝。

一路上有時會遇到紅色的特殊方格，若停在上面則會啟動挑戰模式，需在時限內用Openmoko操縱圓弧，將特定圓圈都圈住，否則又會退回起點。

只要能到達象徵終點的白色方塊，就能得到最終勝利！

二、遊戲操作

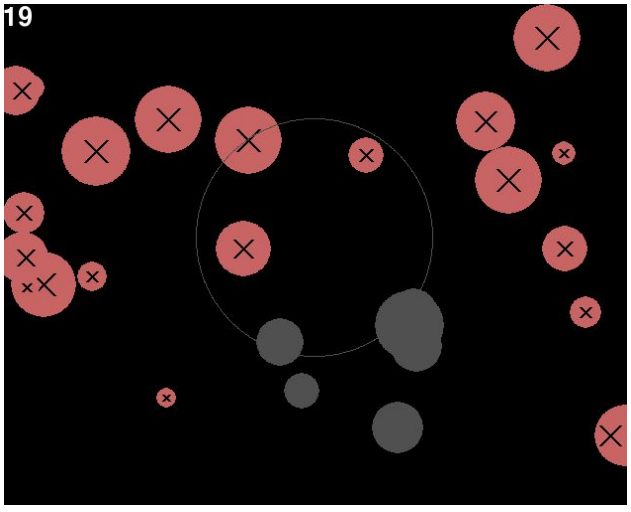
每回合開始前，玩家需投擲骰子，按下f鍵得到行走步數，玩家一直接使用電腦上的鍵盤，玩家二則是使用PCM7230的鍵盤來進行操作。



在遊戲中，雙方使用各自鍵盤上的w, d, s, a進行上右下左的操控，並按下f確定移動，移動的步數必須剛好符合骰子擲出的結果才能前進。雙方輪流移動，以到達白色方格為目標。

灰色方格為牆壁，無法穿越。迷宮中也有許多紅色方格，若停留其上則會觸動挑戰模式，玩家必須完成小遊戲，否則將會退回原點。

點擊Openmoko觸控螢幕以開始小遊戲。



小遊戲的模式，是利用Openmoko操縱圓弧來框住所有的灰色圓圈。玩家需在時限內完成，並不得框住任何紅色圓圈。使用觸控螢幕來確定圈選，若失敗則需退回原點。

Openmoko

Openmoko的操作是分別將手機朝前後左右傾倒來控制圓弧移動；朝向右前、左後傾倒來控制放大縮小。

Up

Down

Left

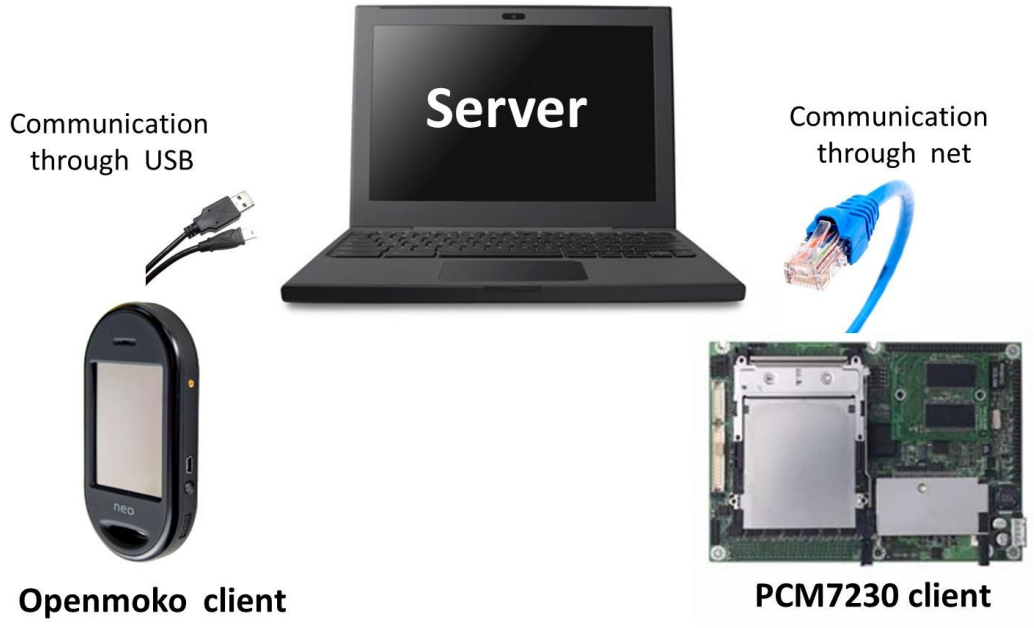
Right

手機背向user:圓弧上移　　手機面向user:圓弧下移　　手機面左:圓弧左移　　手機面右:圓弧右移

手機向右上角傾斜約45度:放大圓弧
手機向左下角傾斜約45度:縮小圓弧

三、系統架構

整個系統由三個主體所構成，包含作為遊戲伺服器的筆電，以及作為控制器的Openmoko和PCM7230。
Openmoko透過USB和伺服器連接，PCM7230則是直接透過網路線連接，伺服器用TCP通訊接收輸入的訊息。



伺服器端分別擁有兩個IP：

- 192.168.1.100 - 設定在網路卡，和PCM7230於同網域。
- 192.168.0.200 - 設定在USB介面，和Openmoko同網域。

以port 2000接收兩者傳來的訊息。

程式

- Openmoko - 執行一個oclient程式，她會偵測觸控螢幕的點擊以及加速器的數值，送出對應訊息。
- PCM7230 - 執行一個client程式，負責偵測鍵盤輸入，並送出對應訊息。
- Server - 執行一個遊戲主程式，她包含一個TCP伺服器，將會接收從兩個客戶端的連線及訊息。

傳送訊息

所有的訊息皆從客戶端傳向伺服器端，資料格式如下：

PCM7230

資料	意義
1 w\n	上
1 d\n	右
1 s\n	下
1 a\n	左
1 f\n	確定

其中，前面的數字代表玩家，可為0或1，這主要用於測試用途，由於僅有第二個玩家使用PCM7230，故在最終版本中，此數字固定為1。

Openmoko

資料	意義
1 om_u\n	上
1 om_r\n	右
1 om_d\n	下

1 om_l\n	左
1 om_b\n	放大
1 om_s\n	縮小
1 om_p\n	點擊

其中的數字僅為格式確認用途，不具意義。

四、實驗過程

構思

為了決定我們期末專題的題目，我們兩人一起討論了許久，最後還是覺得以遊戲的方式呈現最為趣味，而且一邊寫也可以一邊玩。經過一番腦力激盪之後，我們決定大量採用先前實驗所學習到的技術，利用網路訊息連接PCM7230、Openmoko以及電腦來進行遊戲操作。PCM7230可以使用鍵盤輸入，而Openmoko方面則使用加速器以及觸控螢幕。

遊戲的形式，我們決定用走迷宮作為主題，並配合一路上的小遊戲闖關增添趣味。小遊戲的部份我們希望能以Openmoko操作為主，但對於要以什麼形式呈現仍不太確定。後來回家和家人聊天時，想到了這個以圓弧框住圈圈的遊戲，兩人討論過後覺得可行，終於將遊戲內容定案。

Server

遊戲的server端是建置在電腦上，為了便於與Openmoko溝通，使用的是Ubuntu Linux 12.04平台。我們花了不少時間討論遊戲實做的方式。雖然我們先前曾在另一堂課學習過用Verilog配合開發板實做打磚塊遊戲，所以對遊戲設計並不陌生，但是從未直接在PC上用程式實做遊戲。兩人對C語言較為熟悉，但考慮到他的複雜性所以不採用。經過一番研究，最後決定使用Python語言，配合pygame模組[1]，以求敏捷的開發。雖說如此，在遊戲開發過程中，也是閱讀許多教學文件才慢慢理解pygame的使用方式。

整個遊戲程式碼主要分成下列檔案：

- common.py - 程式共用的參數。
- grid.py - 迷宮配置檔，我們共設計了三個迷宮，每次遊戲將會隨機選取其一。
- tcpserver.py - 負責接收Openmoko和PCM7230的訊息並傳送事件給遊戲的伺服器。
- circles.py - 框圈遊戲的場景繪製。
- maze.py - 迷宮繪製，可依各種參數設定繪製出迷宮。
- scene.py - 各個換場、標題的場景繪製。
- amazeing.py - 遊戲主程式，負責處理遊戲邏輯。

tcpserver.py
我們採用ThreadedTCPServer來處理客戶端的連線，可以同時與Openmoko和PCM7230進行連結，同時透過自訂的ThreadedTCPRequestHandler來處理訊息。我們利用makefile()將接受到的訊息當作檔案處理，一行一行透過regex模組來進行比對，若是正確的格式即利用pygame.event.post()將相對應的自訂事件插入pygame的event queue之中。

我們的server設計成可以分別處理不同玩家的訊息，不過由於實際上只有一名玩家使用網路通訊的方式傳送鍵盤事件，所以並未真正使用此功能。

circles.py
我們先建立一個pygame.Surface物件，作為框圈遊戲的畫布。每次當呼叫generate()時就會重新產生新的配置。首先透過random.randint()隨機產生一塊圓形區塊的座標與半徑，緊接著，在圓形區塊內透過

pygame.draw.circle隨機製造數個灰色圓圈、在區塊之外隨機製造紅色圓圈，並紀錄每個圓圈的座標與半徑。generate()同時也會初始化遊戲倒數的秒數，從20到30秒不等。

除此之外，Circles還有下列method可供主程式呼叫，以更新遊戲狀態：

- countdown()可以用來更新倒數秒數，並傳回是否已經結束。他是利用pygame.time.get_ticks()取得目前的時間偏移值，並與傳入的起始時間值做比較，設定兩者相差的秒數。
- is_win()會依據目前圓弧的大小與座標，比對是否框住所有灰色圈圈或是否有框住紅色圈圈，並傳回結果。
- update()會依據目前的參數重新繪製畫布。
- move_direction()會根據傳入的方向，移動圓弧座標。
- enlarge(), compress()則是放大與縮小圓弧。

由於許多因素的作用，遊戲迴圈每次判斷邏輯時所經過的時間並不一定相同，若每次呼叫移動函式都移動固定座標，則可能會有忽快忽慢的效應。故以上三個函式都是根據時間經過的長短乘上預先設定的移動速度來決定最後的偏移值。時間經過數值的取得則是利用pygame Clock object 的 tick()。[2]

主程式可透過get_surface()取得畫布，並繪製在螢幕上。

maze.py
這是繪製迷宮的主要程式，每當主程式要求取得迷宮畫布時，她就會根據目前使用者的位置，繪製整個迷宮的配置。迷宮主要由一個表格所構成，並從grid.py的設定中，隨機讀取任一迷宮配置，每個表格欄位用一個數字代表他的性質。_draw_grids()會根據不同性質而塗上不同的顏色。

此外，對於玩家的移動我們也做了複雜的判斷，使用者必須剛好移動骰子指定的步數apply_moves()才會真正產生作用，且每次移動時不得重複走過同一個格子，這方面的判斷是透過在經過的格子上做記號來達成。（當然，下一次移動可重複走上一次走過的部份，只是在單次移動時不得重複。）

每次在移動時，我們也會在路徑上著色，讓使用者能輕鬆判斷行走的路線。

scene.py
在這個檔案中，我們放置了各個過場、標題的物件。文字方面主要是透過pygame.font.Font物件來設定字型，圖像則是透過pygame.image.load()來載入，骰子的圖片使用Inkscape自行繪製。

amezeing.py
遊戲的主程式，執行start()以後就會不斷迴圈，執行下列函式：

- handle_events() - 將event queue裡的事件一次取出，並判斷按下的按鈕。
- main_loop() - 遊戲邏輯，根據handle_events的結果變更狀態。
- update_display() - 根據目前狀態更新螢幕顯示。

此外，每次迴圈都會執行pygame.time.get_ticks()取得此次迴圈所經過的時間，用來調整框圈遊戲的快慢。

整個程式為一個finite state machine，主要分為以下狀態：

- ST_START - 開場，按下f或觸控螢幕初始化迷宮並進入ST_RANDOM。
- ST_RANDOM - 投擲骰子。
- ST_MOVE -玩家在迷宮中移動。
- ST_GAMEINTRO - 觸發框圈遊戲，按下觸控螢幕開始。
- ST_GAMESTART - 框圈遊戲。
- ST_GAMEPASS - 成功通過框圈遊戲。
- ST_GAMEFAIL - 框圈遊戲失敗畫面。
- ST_WIN - 最終勝利畫面。

main_loop()將會根據目前的狀態以及輸入，呼叫其他模組來更新遊戲狀態。

Openmoko

我們撰寫oclient.c透過usb與server連線以進行通訊，負責傳送g-sensor和touchscreen的資料。我們上網搜尋關於TCP/IP server and client communication的範例[3]，並參考lab7來編寫這部分的程式。

首先，根據lab8 設定連線主機的IP為192.168.0.200，port為2000，而openmoko 本身預設的IP則為192.168.0.202，接著使用socket()來create socket，再用connect() 將socket 連線到server address，之後就可以開始進行讀寫的程序了。

我們以file的方式打開/dev/input/event3來讀取g-sensor的資料，由input_event 的type, code, value等數值判斷目前手機的傾斜狀態，並根據數值大小來判斷手機狀態，若超過一定門檻，即傳送up, down, left, right, big(放大), small(縮小)等等訊息給server。

關於要如何讀取touchscreen 的data，一開始不知從何著手，搜尋網路教學未果，後來想到它應該也是放在/dev/input 的目錄下，利用ssh 登入後果然在該資料夾中找到touchscreen0，參考lab8 的sample code，我們先將訊息印出來以進行分析，但觀察了許久後，仍然對於哪一筆是觸碰到螢幕瞬間的data沒有結論，後來依據網路上找到的程式[4]，才知道pressed和released的type, code, value的值。

我們寫好code並預期應該會傳送g-sensor和touchscreen的資料給server，卻發現client端若是沒有按下觸控螢幕，則會停止讀取g-sensor 的data，檢視程式後了解到是在對touchscreen 作read()的時候被block，後來在程式中加入is_available()，才得以解決此問題，它是運用select() 來判斷在一定時間內是否有讀到值，若沒有的話則會略過繼續執行程式。[5]

我們也用了同樣的技巧來解決快速移動Openmoko時會造成g-sensor輸入block而使程式停擺的問題，所以理論上即使快速移動Openmoko也不會影響觸控螢幕的處理。

由於伺服端會根據遊戲狀態判斷輸入的處理，因此在client不需做狀態的處理，只要盲目的送出訊息即可。程式執行以後就會不斷嘗試與伺服器建立TCP連線，若意外斷線時也會重新連結。我們也取消了SIGPIPE的中斷，讓程式不會因為在斷線時執行write()而造成終止的情形：[6]

```
signal(SIGPIPE, SIG_IGN);
```

PCM7230

PCM7230主要是透過client.c程式，負責第二位使用者的鍵盤輸入，使用到的按鍵分別有 w, d, s, d, f，分別代表上、右、下、左、確定。程式在接受到使用者輸入後就會傳送相對應的訊息至遊戲伺服器。

我們按照lab7的設定方式，將PCM7230的IP設為192.168.1.200，而遊戲伺服器則定為192.168.1.100、port 2000，和Openmoko的client基本上使用同樣的方法建立連線。為了便於遊戲的進行，我們將PCM7230設定為一開機就會自動連線。首先修改 /etc/network/interfaces，加入以下數行，設定開機時自動啟用eth0並設定IP位置：

```
auto eth0
iface eth0 inet static
        address 192.168.1.200
        netmask 255.255.255.0
        gateway 192.168.1.100
```

緊接著，將client程式放入rootfs的/root/資料夾，並建立/root/.profile檔案，加入：

```
/root/client
```

執行chmod +x .profile指令，啟用執行權限，讓她開機便自動執行client程式。

在接收使用者輸入方面，主要需要解決的問題有兩個：

1.預設的輸入模式只有在使用者輸入Enter以後，程式才有辦法得到輸入字元。

2.若使用者按住按鍵不放，則程式會不斷得到重複字元，我們希望可以避免此行為。

第一個問題，透過從lab1學習到的tcsetattr()將ICANON標誌取消，進入noncanonical mode之後就能即時得到使用者的輸入。[7]

第二個問題，我們則是透過select()，可以在一定時間內就得知是否已經有輸入字元，藉此我們得以檢測是否在短時間內輸入同樣字元，若是則忽略輸入，此方法雖然不完全精確，但確實能解決我們的問題。

由於伺服端會根據遊戲狀態判斷輸入的處理，因此在client不需做狀態的處理，只要盲目的送出訊息即可。程式執行以後就會不斷嘗試與伺服器建立TCP連線，若意外斷線時也會重新連結。我們也取消了SIGPIPE的中斷，讓程式不會因為在斷線時執行write()而造成終止的情形。

一開始測試的時候還算順利，但後來網路功能突然無法正常運作，經過一段時間的測試。才發現似乎是線本身的問題，只有在將線調成特定角度才能連結，應該是有接觸不良的現象產生，還好跟助教聯繫後得以更換成新的線，順利解決此問題。

Reference

1. http://en.wikipedia.org/wiki/Pygame
2. http://stackoverflow.com/questions/8252860/
3. http://www.linuxhowtos.org/C_C++/socket.htm
4. http://downloads.openmoko.org/developer/sources/svn/openmoko-panel-mainmenu/openmoko-panel-mainmenu/src/buttonactions.c
5. http://linux.die.net/man/2/select
6. http://stackoverflow.com/questions/108183/
7. http://linux.die.net/man/3/termios

四、心得

這次我們經過了許多討論才決定最後遊戲的面貌，在這次專題實做的過程中，我們使用了許多先前學到的技巧，但也研究了新的作法。對於觸控式螢幕和g-sensor的輸入以及事件的判斷我們花了不少心思。而在兩個client端，我們同樣遇到輸入block住的問題，並透過select()成功解決。原本以為在框圈遊戲中，client端和server端的網路通訊可能需要花不少時間調整訊息的速度，然而測試的結果卻非常順利。server端的實做，利用了從未使用過的pygame模組，成功完成了作品。在過程中我們常常為了解決問題而去學習特定的工具與作法，這和從前先學東西再拿來用的習慣真的很不同。慢慢完成每一片拼圖後，最後的結合非常順利，雖然遇到了網路的小問題，可是也順利找到原因。