LabVIEW 期末專案 - 簡易音樂遊戲

111550037 嚴偉哲　　111550109 黃琮仁

一、 動機

我們兩人平時都有玩音樂遊戲的習慣，因此想藉這次機會來嘗試製作並推廣這類型的遊戲。

二、 簡介

音樂遊戲(Rhythm Game)是一種遊戲類型，基本元素為**歌曲**及其對應的**譜面**。在遊玩過程中，不同形式的鍵(note)會根據譜面下落至判定線，而玩家須在該鍵移到判定線的時候進行點擊。

[圖 1] 音樂遊戲(太鼓達人)遊玩畫面

三、 需求分析

1| 輸入方式

為純鍵盤輸入，用**左右鍵**選擇，Enter 確認，D, F 及 J, K 分別點擊上、下兩軌，Esc 為關閉遊戲。

2| 素材

預計使用**兩首歌**來做遊戲，並要再找**曲繪**(一首歌的封面圖)及**背景圖**來美化介面。整體畫面呈現而言，我們打算完全使用 2D Picture 來製作。譜面部分，需要自己設計出一套文字格式以便匯入遊戲。

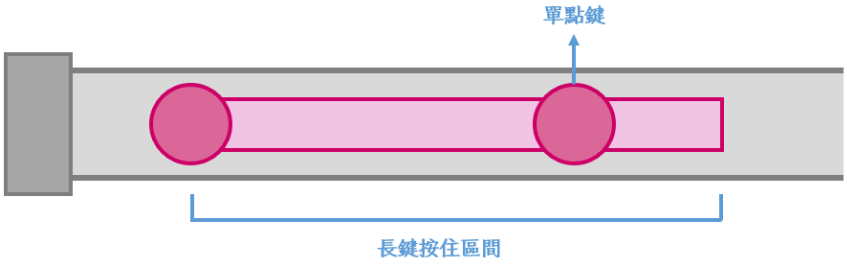
3I　遊玩方式

在遊戲中，玩家須在對的時機點擊上、下兩個軌道的鍵，點擊後會顯示出該鍵的準確度並更新分數。其中鍵分為二種類別：

單點(tap) – 僅須在鍵到達判定線時按一下即可。

長按(hold) – 點擊後要按到鍵尾完全通過判定鍵。

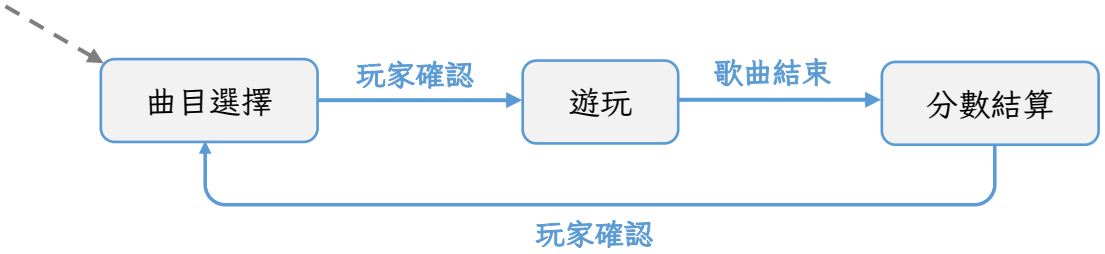
有時，長鍵還沒結束時會跟著單點鍵，這時需要用另一個按鍵來點擊。(例：以[圖 2]來說，假設長鍵是用 F 按，接下來的單點鍵要用 D 按)



[圖 2]譜面特殊配置

四、　程式規劃

遊戲流程主要可分成三階段 – **曲目選擇**、**遊玩**、**分數結算**。而會有如[表 1]的狀態圖：



[表 1]遊戲狀態圖

我們可以將此表以**狀態機**的形式於遊戲中實現。在切換狀態的時候，會需要做**轉場過渡**，因此也要另外寫一套偵測此事件的子程式。

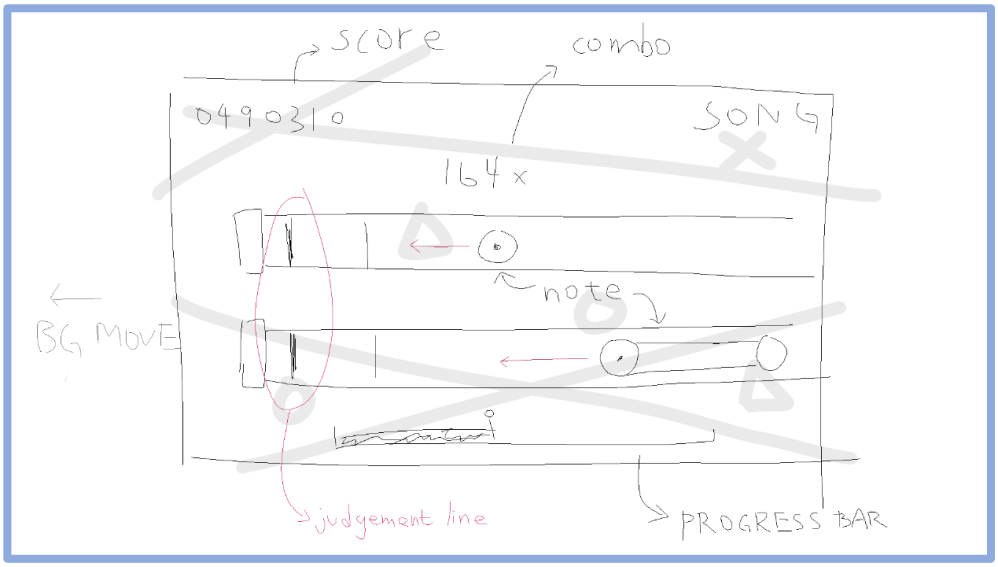
1I　階段規劃

在曲目選擇階段，遊戲要儲存玩家選擇的**歌曲**、**難度**，以及**展示模式**(即 AI)，並供遊玩階段時使用。(圖 3)

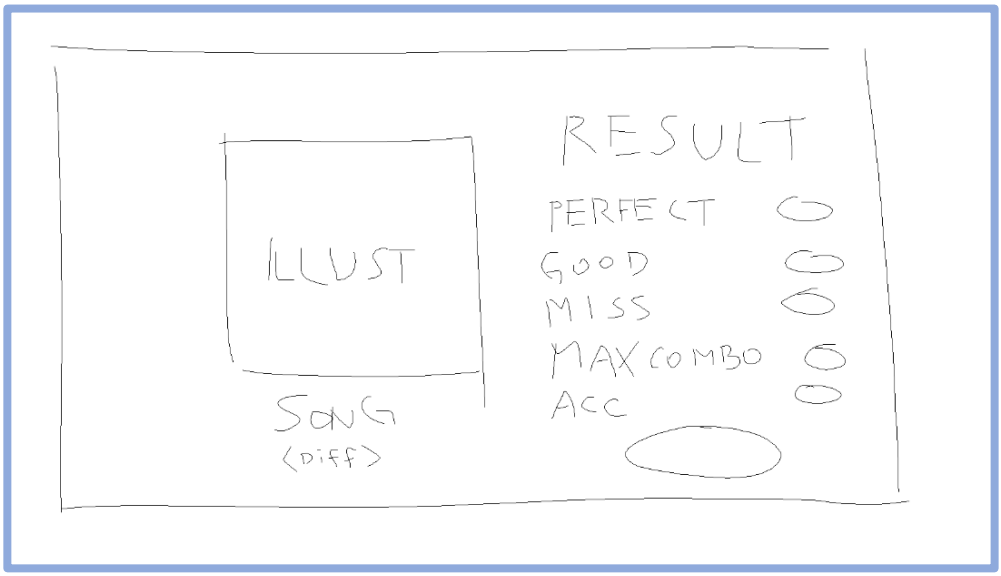
全部確認完畢後，遊戲會根據曲目其難度讀取相對應的譜面，接著要做到鍵的**下落動畫**，以及**準度判定**的元素。資料儲存方面，須包含每個鍵的**準度加總**和**最大連擊次數**。此外，因為有加入長鍵玩法，要額外儲存玩家按長鍵時是使用哪個鍵，以便後續處理**鬆鍵判定**。顯示上除譜面之外，亦會加上**目前分數**、**遊玩進度**和**曲目名稱**。(圖 4)

分數結算時，會把準度統計、最大連擊、總分、曲目及難度標示出來，並等待玩家回到主選單。(圖 5)

[圖 3]曲目選擇介面草圖



[圖 4]遊玩介面草圖



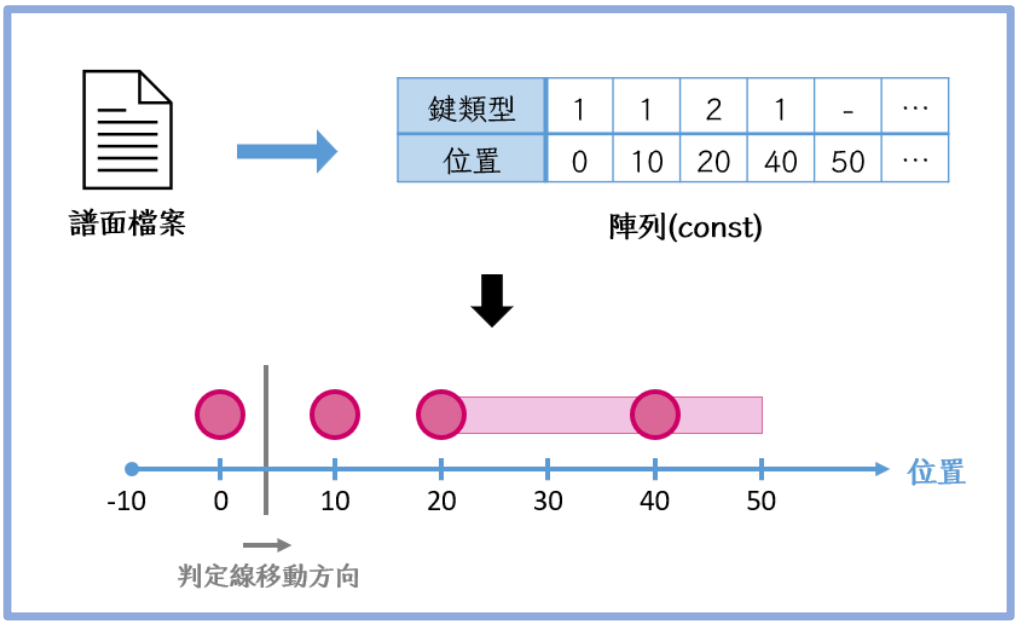
[圖 5]分數結算介面草圖

2| 譜面顯示

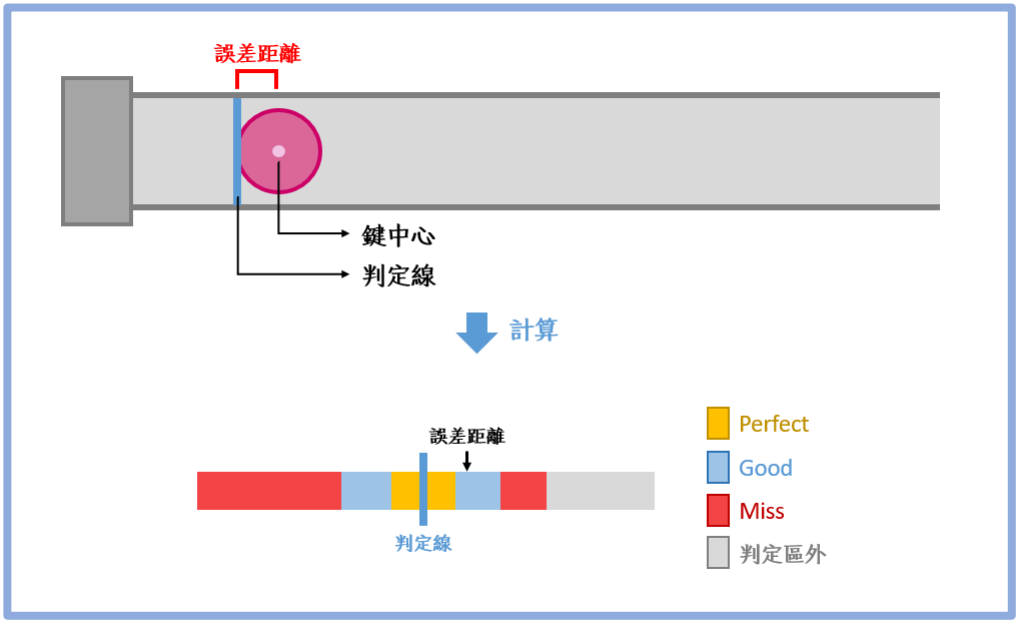
遊戲在讀取譜面時，會將各個鍵的位置計算並儲存於一**陣列**中，另外會有一記錄判定線位置的變數。遊玩時，判定線的位置會隨時間增加，而遊戲會渲染出鍵與判定線的**相對位置**。(圖 6)

3| 準度判定

首先，程式會根據譜面下落速度及預先設定的判定時間表計算出判定距離。接著，玩家在點擊一按鍵後，遊戲會依當下鍵跟判定線的距離決定精準度。(圖 7)



[圖 6]譜面顯示方式構想



[圖 7]準度判定說明

五、 遊戲製作

這部分前半段會說明遊戲功能面向是如何實現，後半段為視覺面向。

1| 譜面格式

譜面設計為我們自己**親手編寫**，不使用隨機生成譜面的主要原因是**可玩性**，電腦分辨不出細微的旋律以及該取那些音色來寫譜，因此就算生成得出來，譜面也多半無法襯托當下音樂的起伏。而人工撰寫的譜可以經由多次測試，修改調整成大多數玩家認為好玩的關卡。當然，固定式的譜面玩久了也無可避免的會感到無聊，這也是許多音樂遊戲持續收錄新歌的原因。

格式上有些微參考 BMS(Be-Music Source)格式(圖 8)，算是最早期的音樂遊戲譜面格式之一。

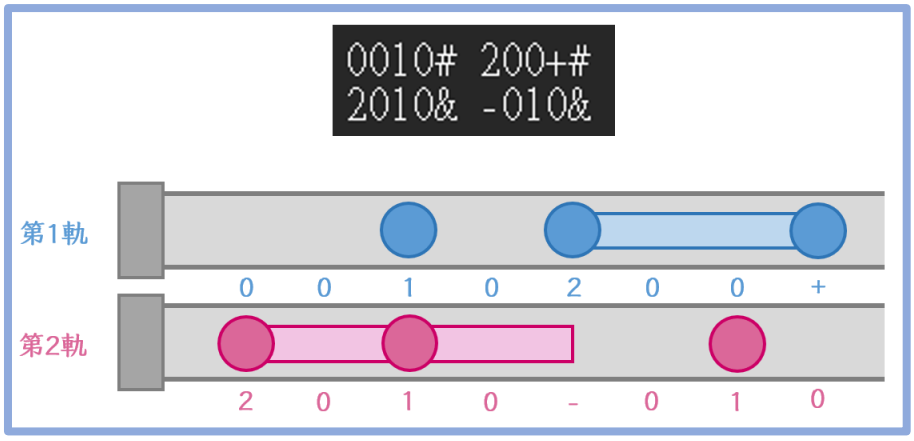
```
*---------------------- HEADER FIELD
#PLAYER 1
#GENRE Sample
#TITLE Sample
#ARTIST Sample
#BPM 120
#PLAYLEVEL 5
#TOTAL 100
#RANK 2
#bmp00 miss.bmp
#bmp01 1.bmp
#wav01 1.wav
*---------------------- MAIN DATA FIELD
#00111:01010101
#00211:0101010001010100
```

[圖 8] BMS 檔案範例

為求修改方便，我們使用了 txt **檔案**來設計格式，主要分兩部分 – **標頭**以及譜面。標頭包含**曲名**、BPM 以及**時長**，並以逗號分隔。譜面則是以 1 拍為單位，每拍用"**#**"(第 1 軌)或"**&**"(第 2 軌)分隔，4 拍為一行，二軌交互編寫。[表 2]說明了各種鍵型的表示方式，而[圖 9]則為文字與實際譜面的對照。

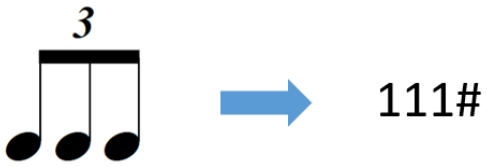
記號	代表鍵型
0	空格/無鍵
1	單點鍵
2	長鍵
-	長鍵結束
+	長鍵結束位置有單點鍵

[表 2]鍵型記號



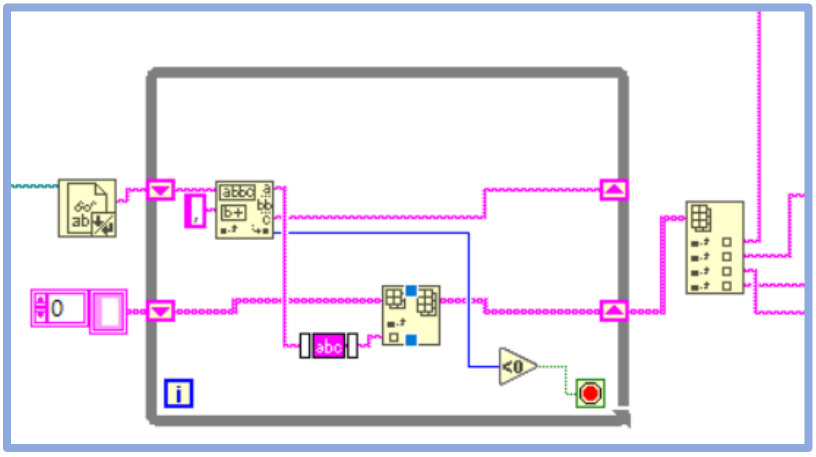
[圖 9]譜面對照

在每拍中，記號的數量就代表著鍵的**密集度**，或可稱為**連音**，這意味著寫譜時能寫出三連音等特殊旋律。



[圖 10]三連音寫譜範例

在讀取檔案時，遊戲會先將資料根據逗號拆成四個部分 – 曲名、BPM、時長以及譜面。(圖 11)

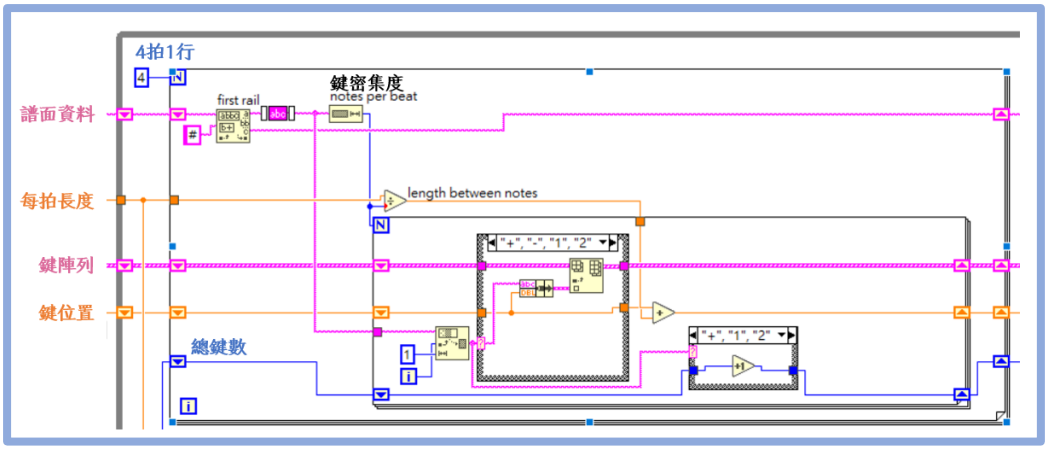


[圖 11]譜面讀取程式

接著，遊戲會根據 BPM 和給定的「每拍長度」換算下落速度，公式如下：

下落速度 = (BPM / 60)*每拍長度

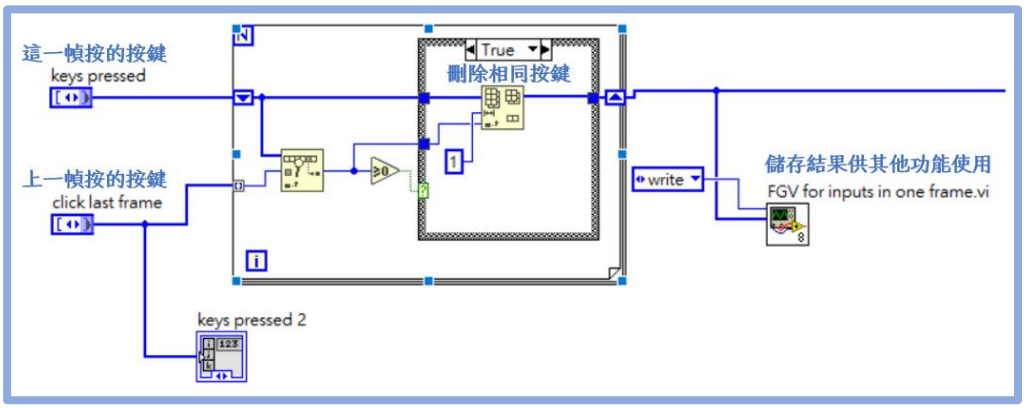
同時，譜面被轉換成陣列，儲存所有不是 0 的鍵型，並依其出現時間換算成位置。(圖 12)



[圖 12]譜面轉換陣列程式

2I 點擊與判定

原先的想法是使用 Event Structure 內的 **Scan Code** 直接偵測按鍵，但因為 Scan Code 只回傳 1 個鍵的值，在玩家**同時按下兩個鍵**時其中一鍵可能會被忽略導致遊戲體驗不佳，所以我們採用會回傳按鍵陣列的 **Acquire Input Data**。為實現與 Key Down 同樣功能的程式，我們將按鍵陣列接上 Shift Register 並比較二次迴圈間的差異。(圖 13)



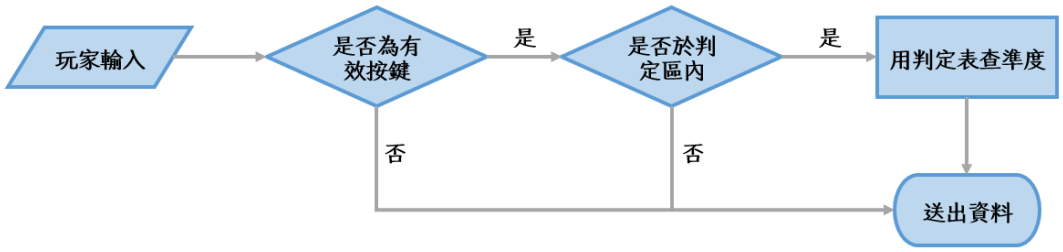
[圖 13]找出這一幀按的按鍵

為了讓遊戲知道玩家點擊時是想按哪個鍵，遊戲會記著**下一個未判定的鍵的陣列編號**(目標鍵)，並以此鍵位置作誤差距離計算的標準。而實際使用的判定區間是參考 *Arcaea* 的判定，如[表 3]所示。

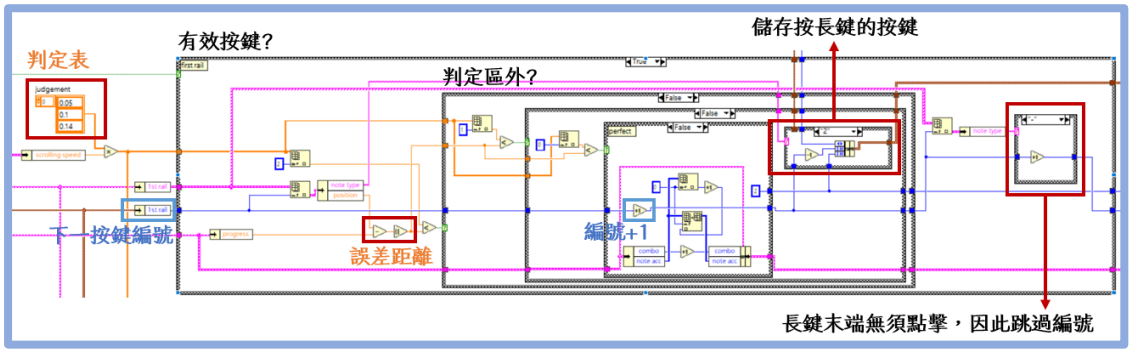
準度	判定區間
Perfect	≤50ms
Good	≤100ms
Miss	>100ms (未到達判定線時為≤140ms)

[表 3]判定區間

[表 3]乘上上一部份得出的下落速度即可獲得距離版本的判定區間。接著，遊戲會根據上述資訊算出準度，過程可以[表 4]及[圖 14]描述。

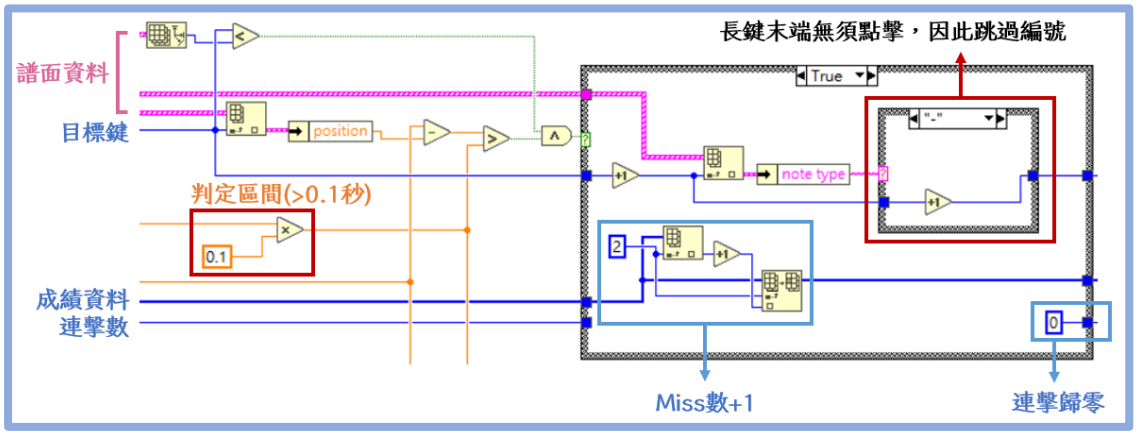


[表 4]判定流程圖



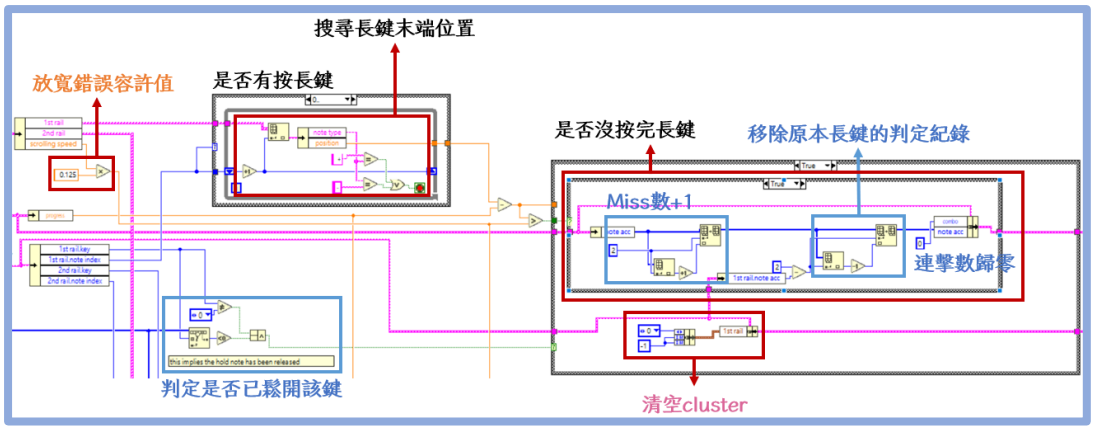
[圖 14]點擊判定程式

此外，若玩家**未點擊**，遊戲也要算成 Miss，這可以用**目標鍵的誤差距離**做判定。(圖 15)

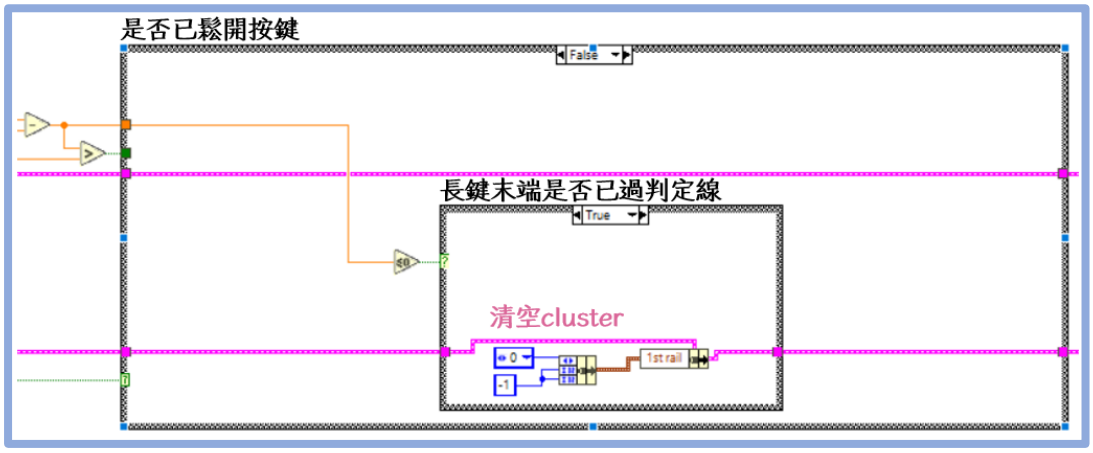


[圖 15]玩家未點擊之處理程式

若點擊的鍵為長鍵，遊戲還需檢查玩家是否**過早放開**，因此我們加了一個 cluster 來儲存**按長鍵的按鍵**以及**長鍵的編號**。在過程中，遊戲會不斷檢查按下的鍵是否放開，假設為真會根據編號找出長鍵末端位置，並依此判斷玩家是否按完。此外，若長鍵結束後玩家仍按著該按鍵，遊戲需自動**將 cluster 清空**(即幫玩家放開按鍵)。(圖 16)

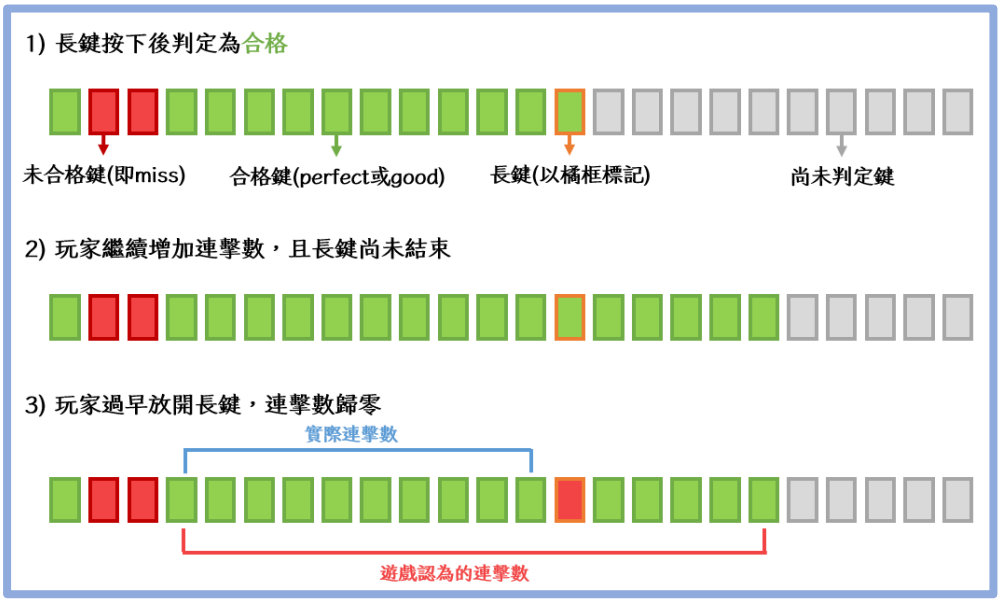


[圖 16.1]鬆鍵判定程式



[圖 16.2]未鬆鍵判定程式

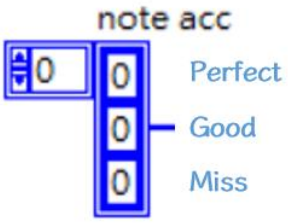
不過，這樣子的設計有個缺點 – 這裡我們處理中斷連擊數的方式僅僅為將其歸零，並未調整最大連擊數而導致**其數值可能比實際情況大**，如[圖 17]所示。這會在<未來發展>的部分討論修改方式。



[圖 17]錯誤判定示意圖

3| 分數計算

在遊玩過程中，遊戲會紀錄玩家點擊鍵的準度，儲存在一長度 3 的陣列中(圖 18)。



[圖 18]鍵準度陣列

在計算分數時，遊戲會考慮二項元素 – **準度**以及**連擊比例**，各佔 900,000 分以及 100,000 分，因此總分為 1,000,000 分。公式如下所示：

準度 = (Perfect 數 + 0.6*Good 數) / 總鍵數
連擊比例 = 最大連擊 / 總鍵數

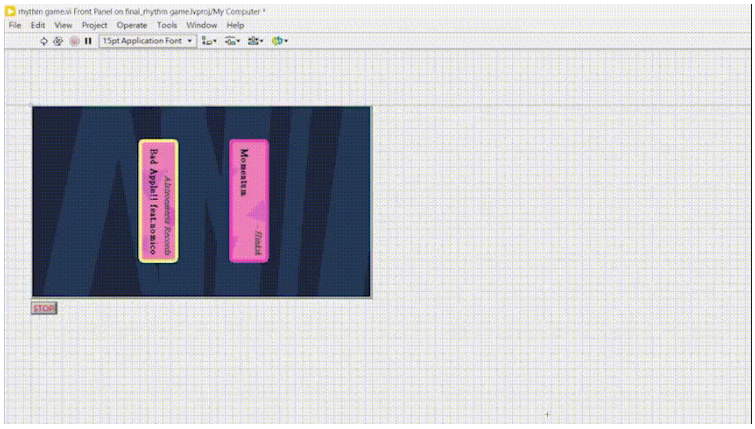
總分 = 900,000 * 準度 + 100,000 * 連擊比例

計算完成後，總分會顯示在遊玩畫面的左上角以及結算畫面的右下角。

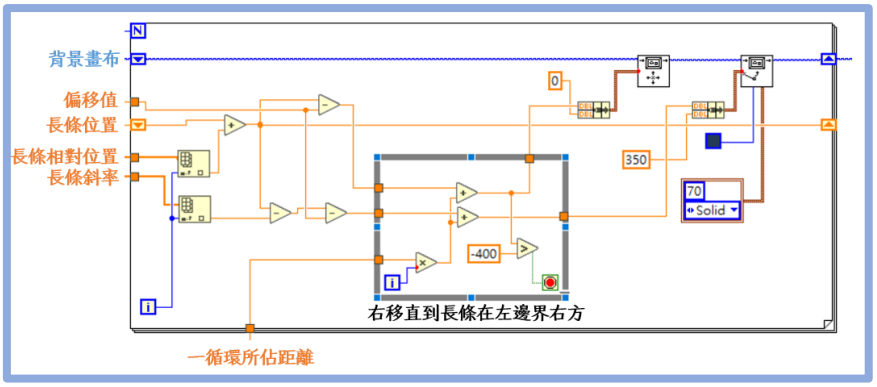
4| 遊戲渲染

在一開始構思介面的時候，就有想把畫面做成有**透視感**的想法。為了達成此目的，以及方便修改調整，我們使用了二個 2D Picture 畫布當作**前景**和**背景**使用，其中前景預設顏色調整為**全透明**。(影 1)

背景部分是由許多向右移動的長條構成，在剛啟動遊戲時以**隨機**生成 20 組長條**相對位置**及**斜率**，並拆成兩份以不同速率移動。為了節省資源，程式會將離開畫面左側的長條平移至畫面右側(圖 19)。而渲染結果同樣呈現於[影 1]中。

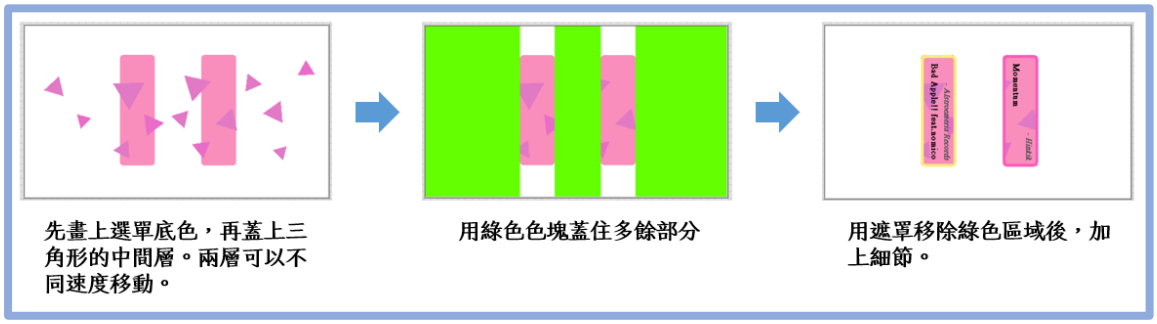


[影 1]雙圖層展示



[圖 19]背景渲染程式

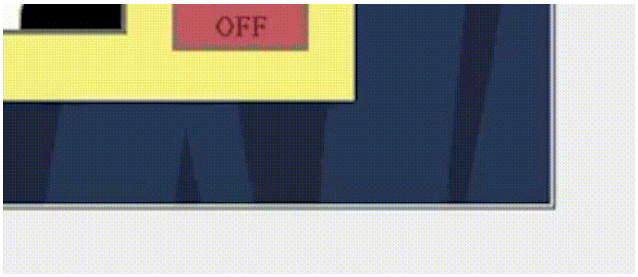
在曲目選擇界面設計上，我們想讓選單的填充背景(按鈕內的深色三角形)也具有透視感，因此使用了 LabVIEW 內提供的**遮罩**來實現（圖 20），完成後的樣子可於[影 2]中看見。



[圖 20]遮罩原理說明

[影 2]選單透視效果展示

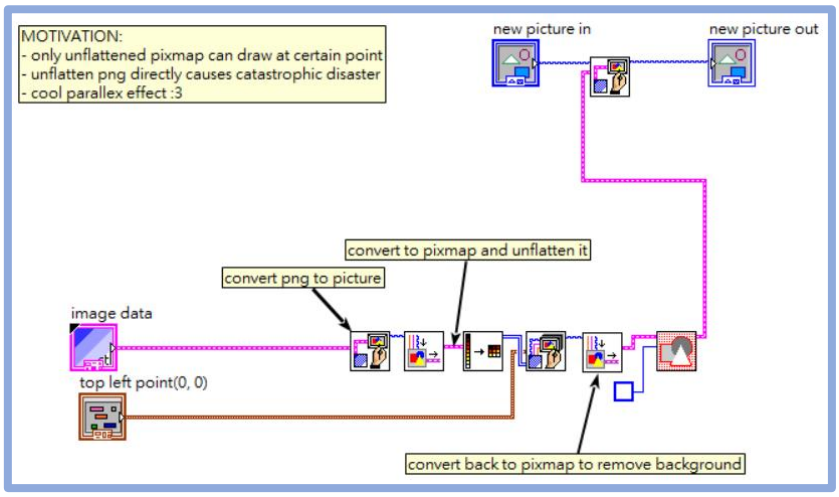
在許多地方我們都利用了這個技術達到基礎函式無法實現的效果，如[影 2]中有展示的**選單展開**，和過場畫面中把**文字部分裁切**的效果。(影 3)



[影 3]過場動畫

在導入圖片素材的時候，我們發現兩件重要的問題。首先是 LabVIEW **繪製外部圖片**的速度偏慢，這在曲目選擇及分數結算畫面中可以明顯地看到類似「**掉幀**」的現象，而掉幀在許多遊戲，特別是音樂遊戲這種講求**高度精準**的類型，是無法接受的。為此，我們也嘗試了把圖片匯入後儲存下來等方法，但最後決定在**遊玩畫面**的所有物件，皆使用 LabVIEW 提供的**基本繪圖函式**製作。

第二點是圖片匯入後，是用 Flattened Pixmap 的形式做資料傳遞，若要實現**在畫布的不同點把圖片畫上去**，需要先轉成 Unflattened Pixmap。但是在實作時，我們發現直接轉換會造成**圖片毀損**，因此需要先畫在 2D Picture 上，再做後續的處理。(圖 21)



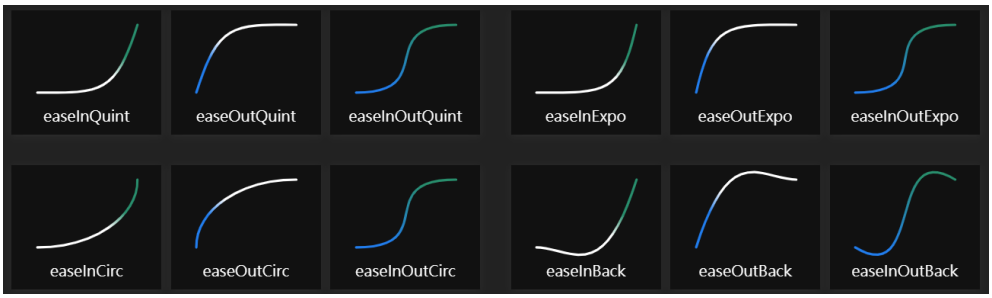
[圖 21]繪製外部圖片至指定點上程式

5I 動畫元素

LabVIEW 本身並無除改變座標外的移動功能，但仍可用**計時器**搭配 FGV 及**緩動函數(Easing Function)**來實現。

首先，主程式會設置一個全域計時器，紀錄自啟動遊戲起的**毫秒數**。這個計時器會控制動畫、鍵下落及判斷音樂是否結束，而因為它是以毫秒計不是以迴圈數計，就算掉幀也不會讓音樂和鍵對不上，也就完美解決音樂遊戲最重要的部分，即「**玩家跟著音樂打擊**」。

緩動函數會根據給定時間(0≤t≤1)換算出位置(0≤d≤1)，因為其函數不是簡單的線性方程式，物件的移動軌跡會**更加自然**(圖 22)。

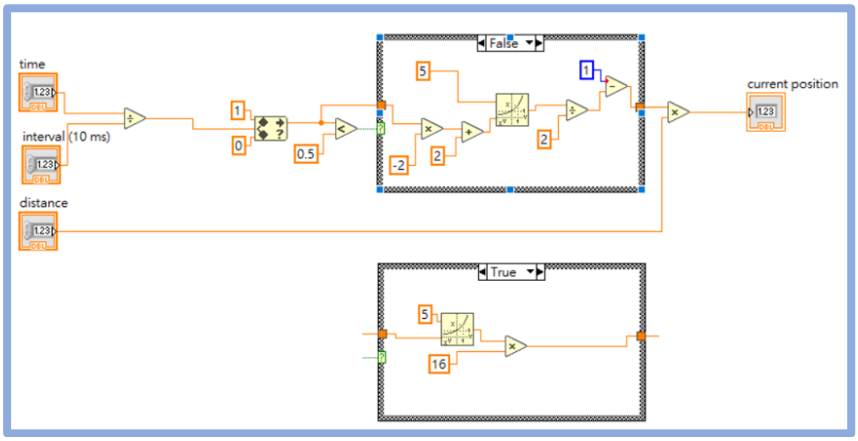


[圖 22]緩動函數圖形範例

我們接著可以拉伸 t 跟 d 的長度來滿足不同需求，舉例來說，easeInOutQuint 的函數為：

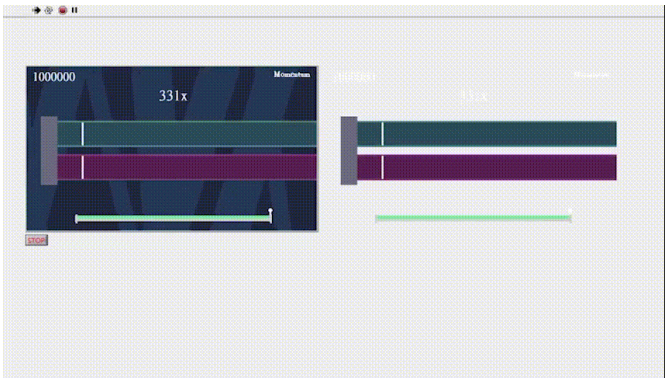
$$f(t) = \begin{cases} 0 & t < 0 \\ 16t^5 & 0 \le t < 0.5 \\ 1 - \dfrac{(-2t+2)^5}{2} & 0.5 \le t < 1 \\ 1 & t \ge 1 \end{cases}$$

若用程式表達會變成[圖 23]的形式。這張圖的 t 是 interval，d 是 distance，而計時器(time)的值會持續增加並回傳目前位置(current position)。

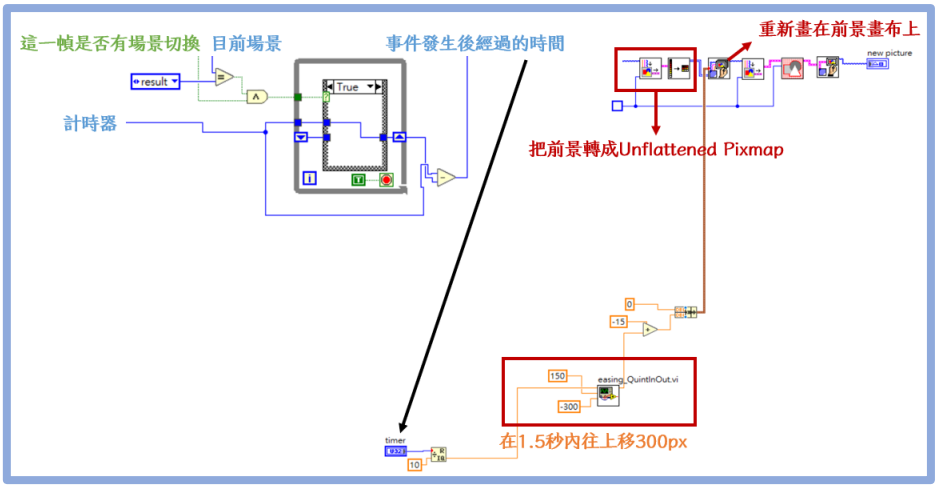


[圖 23]easeInOutQuint 函式

當特定事件發生時，如場景切換、按鍵點擊等，遊戲會使用 FGV 儲存當下的時間並依此作為**動畫時間起點**。像是在遊戲結束後出現的結算畫面(影 4)，是用[圖 23]的緩動函數達成，其觸發條件為**計時器秒數大於音樂時長**。(圖 24)



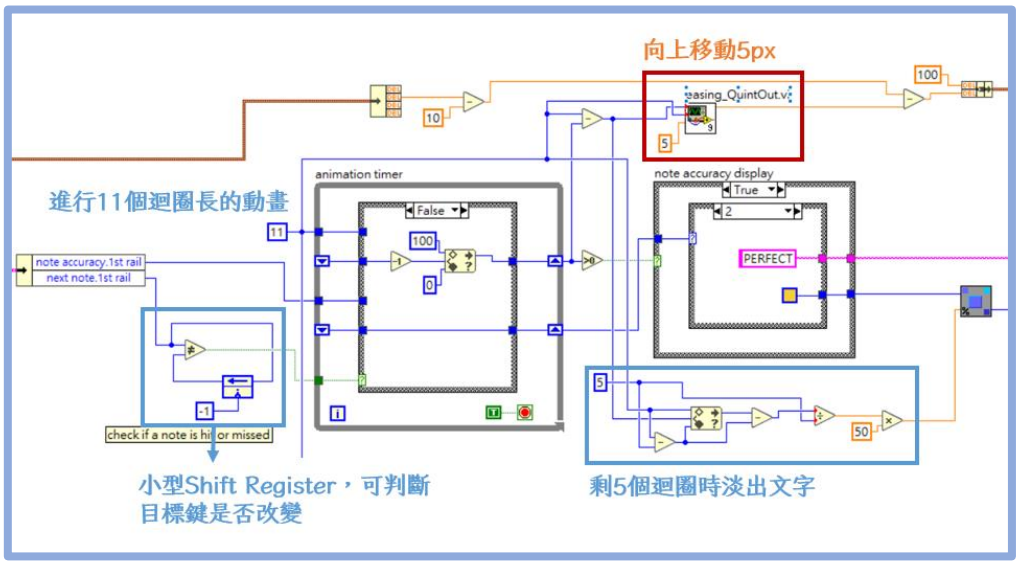
[影 4]結算畫面動畫，右半邊展示了遊戲畫面外的部分



[圖 24]結算畫面動畫程式

在遊戲過程中的各種小動畫(連擊顯示、準度顯示及按鍵點擊特效)因為並**無時間急迫性**，並且為了節省資源**增加流暢度**，是根據**迴圈次數**來進行動畫。以準度顯示為例(影 5)，該動畫會在**目標鍵改變**時播放，較為麻煩的是，LabVIEW 並不支援**透明度**，因此我們採用將**文字顏色變暗**的方式來模擬淡出效果。(圖 25)

[影 5]準度顯示動畫

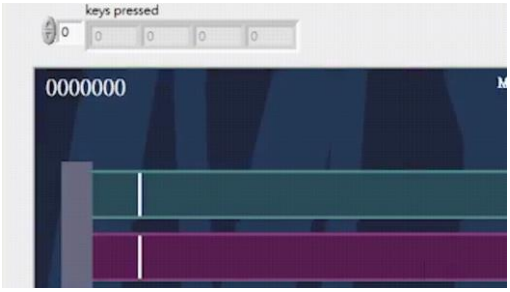


[圖 25]準度顯示動畫程式

61 遊玩體驗

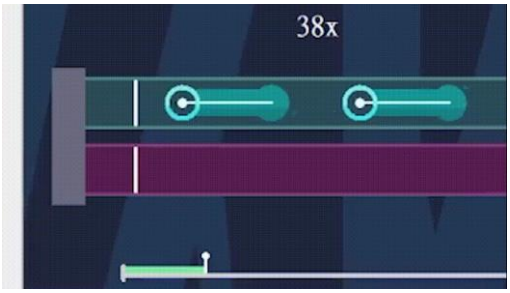
這部分會說明一些非必要，但提升玩家遊玩體驗的程式功能。首先是**容錯**，在[圖 16.1]有提到我們把長鍵的鬆開判定調成早 **0.125 秒**也算通過，這是考慮到玩家在玩較複雜的譜面時可能會提早放手去準備點接下來的鍵。

再來是視覺上的**回饋**，也就是遊戲要放上一些特效如動畫、改變亮度來讓玩家知道遊戲有接收到他們的輸入。在我們的遊戲中，當玩家按下按鍵，相對應的軌道會**亮起**(影 6)，而這裡利用到了[圖 13]存下來的資訊做事件檢測。



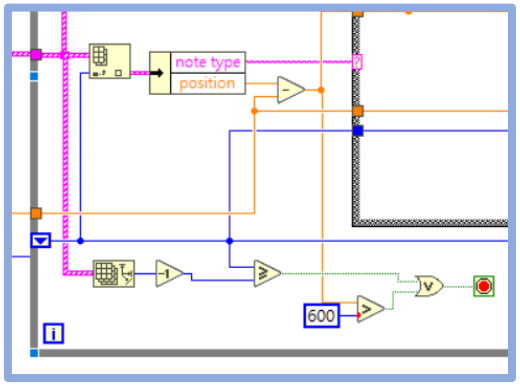
[影 6]軌道特效演示

在玩家點擊一個鍵之後，遊戲會直接讓它消失，但若將相同方法套用在長鍵上會略顯奇怪，因此我們讓還按著的長鍵從判定線渲染到鍵末端。(影 7)



[影 7]長鍵按住時之渲染

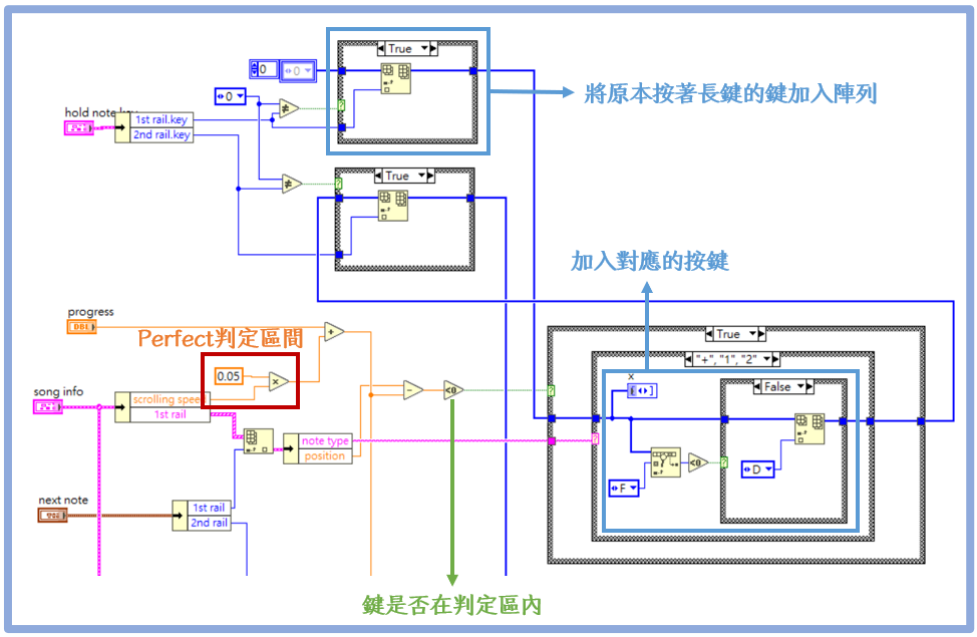
接著是遊戲**順暢性**，除了先前所提到的僅用內建函式繪製物件外，遊戲僅會渲染在畫面內的鍵，在[圖 26]可看到，渲染用的迴圈終止條件是**位置>600 或渲染到陣列末端**。



[圖 26]鍵渲染終止條件

六、 AI 設計

相較於其他遊戲，音樂遊戲的 AI 比較簡單直覺，大致方向就是寫出一個零失誤的玩家。在每次迴圈，AI 會接收到**目前進度**、**目標鍵**跟**譜面資訊**的資料，而 AI 會根據這些資訊計算出與目標鍵的**誤差距離**，若在 Perfect 判定區內則輸出對應的**按鍵陣列**來模擬玩家輸入。此外，若遇到[圖 2]的特殊配置，AI 會輸出含兩個鍵的陣列。[圖 27]展示了該程式的截圖。

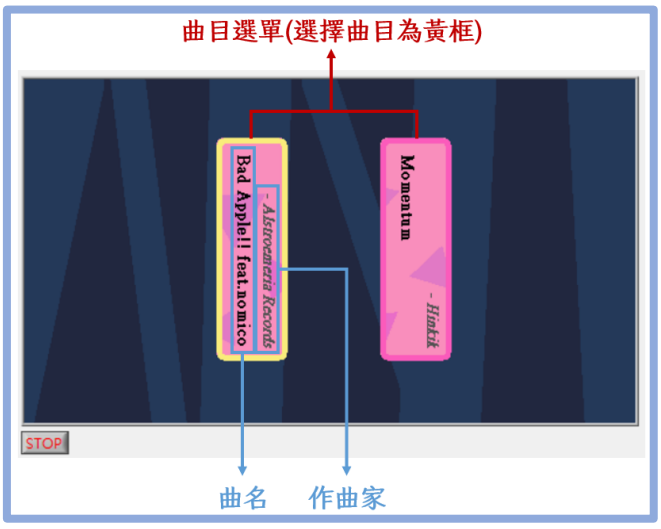


[圖 27]AI 演算法程式

其餘就是將按鍵陣列傳給遊戲做準度判定，大致與玩家操作時相同。

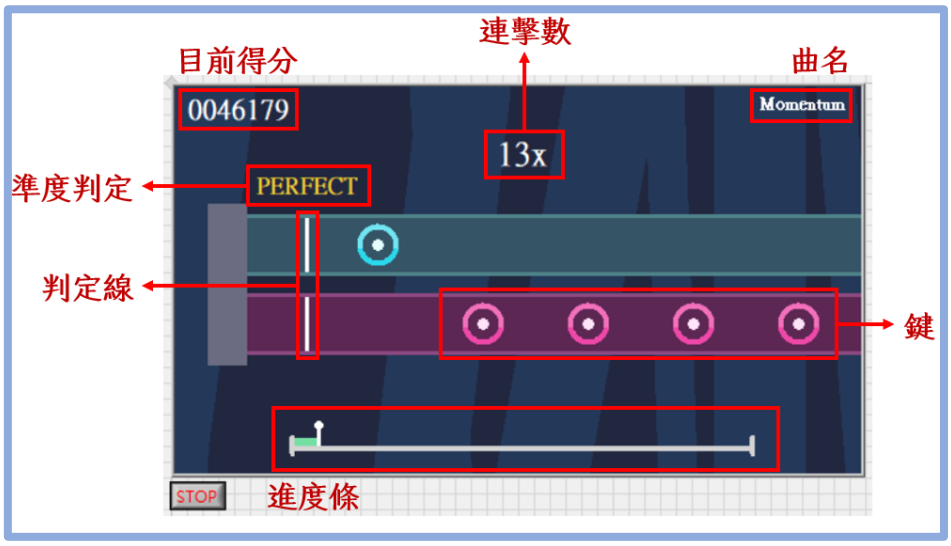
七、 遊戲圖片與影片

[圖 28]到[圖 31]為遊戲截圖並附上說明文字。



[圖 28]曲目選擇介面

[圖 29]難度選擇介面



[圖 30]遊玩介面

[圖 31]分數結算介面

以下為兩個遊戲展示的影片連結，前者為**玩家遊玩**，後者為 **AI 遊玩**。

玩家：https://youtu.be/YGYGrdy6cf8

AI：https://youtu.be/zC_gpSn-PU0

八、 未來發展

這個遊戲還有非常多的潛力以及能改進的地方，礙於時間問題無法及時實現，因此在這部分會討論未來能加入的元素。

1| 錯誤修復

在[圖 17]有提到，**最大連擊數**會被長鍵這個元素所誤判，這裡我們想到了二個解決方案。一是額外創建一個陣列來一一儲存每個鍵的準確度，並於每次更新時找出**最長連續合格數**，並以此作為最大連擊數。但這會大幅增加電腦的運算量，導致可能發生的掉幀問題。

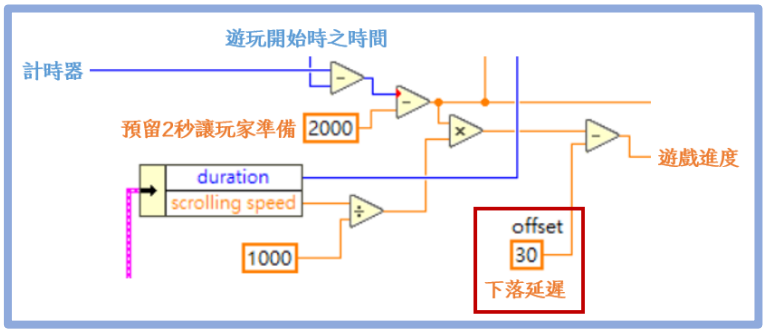
二是將長鍵的準度判定等到鬆鍵或按完再新增至分數表裡，而這會是比較好的解法。

第二個問題是判定上的延遲，在我們的設計中，猜測是因為使用 FGV 或 Shift Register 的緣故，按下按鍵後會**晚一幀**才開始執行操作。這其實不是太嚴重的問題，但我們發現在較低性能的設備中 AI 有時會玩出 Good 的判定。這可以用優化程式碼的方式嘗試解決，例如進一步減少渲染物件。

2| 完善功能

考量到時間不足，我們並未將**回到主選單**和**按鍵教學**的功能做出來，這是以後會優先加入的元素。

除此之外，**下落延遲**也是很重要的一環，因為不同玩家與不同設備對於「鍵對齊音樂」的認知都有些微差距。在目前的遊戲中是晚 30px 下落(圖 32)，在未來是希望以毫秒為單位讓玩家自己調整。



[圖 32]下落延遲與遊戲進度處理程式

還有例如**遊戲暫停**、**可調式下落速度**、**調整音量**、**練習模式**等能讓玩家更**自由選擇**他們玩遊戲的方式的功能，都是希望能加入的東西。

3| 新增玩法

對於一個期末專案來說，2 首歌 4 個譜以及 2 種鍵型已經非常足夠，但未來若要再進一步開發此遊戲，除了加入更多首歌之外，我們認為加入更多鍵型會讓遊戲元素更加豐富，例如**連打鍵**、**不能點**的鍵，或到判定線

前會漸漸**消失**的鍵。再更高難度上，能讓不同鍵有**不同下落速度**，甚至讓整個**軌道移動**，還有加入**血條**元素(失誤時會扣血，歸零時遊戲結束)，都能提升遊戲整體的可玩性。

4. 畫面/音效調整

我們也考慮增加**打擊音效**，即遊戲會在玩家點擊鍵的時候播放鈸或鈴鼓的音效，這是為了讓玩家更有效率地確認節奏。此外，也想在遊玩畫面時有**不同背景**，以及連擊數到一定數量時**改變背景顏色**。

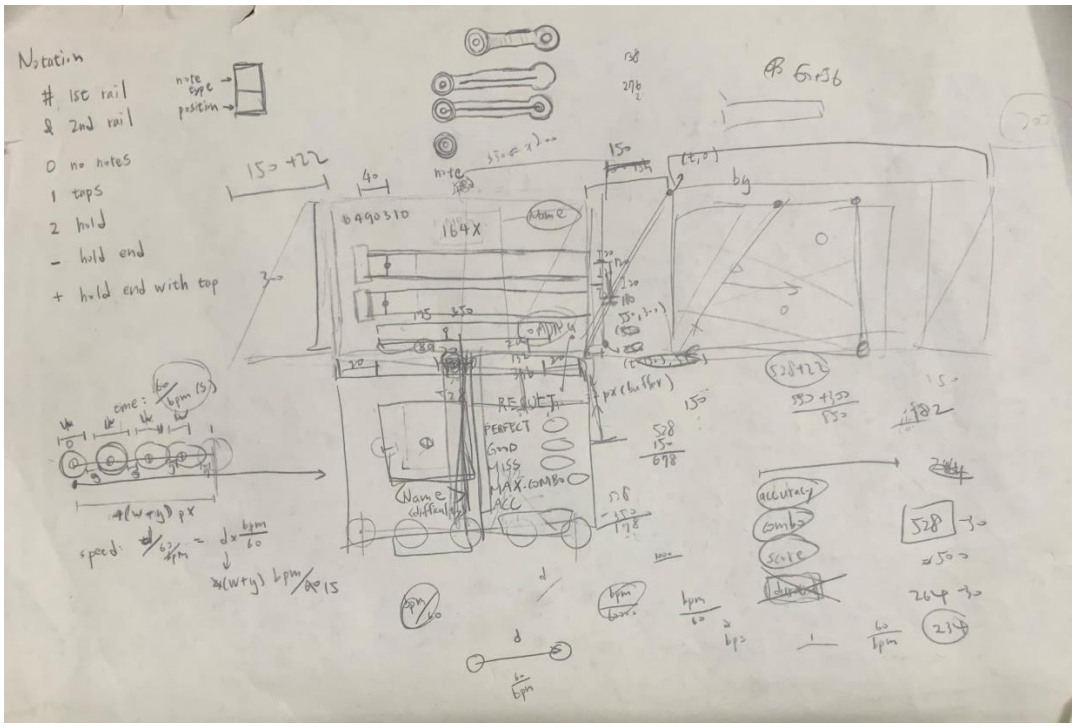
在**畫面偏小**的問題上，因為當初設計時為了除錯/修改方便，想讓畫布只佔螢幕一半以便同時查看 Block Diagram 及 Front Panel，所以忘記調整回來。對此，可以將整個畫布轉成 Pixmap 再用演算法放大，或手動更改每一個物件的大小及位置。

5. 多人連線

這是我們來不及做的專案要求部分，若要實現，預計是會將**其他玩家的分數及血條**放在側邊來比誰的分數高或誰能玩完整首歌。

九、 參考資料及素材使用

遊戲玩法：Muse Dash
介面設計：太鼓達人、Muse Dash、Phigros
判定：Arcaea
分數計算：Phigros
音樂：Hinkik - Momentum、
Alstroemeria Records - Bad Apple!! feat.nomico
參考資料：https://easings.net/#
https://en.wikipedia.org/wiki/Be-Music_Source
https://phigros.fandom.com/wiki/Game_Mechanics
https://arcaea.fandom.com/wiki/Scoring#Timing



[圖 33]設計遊戲時的規畫與概念圖