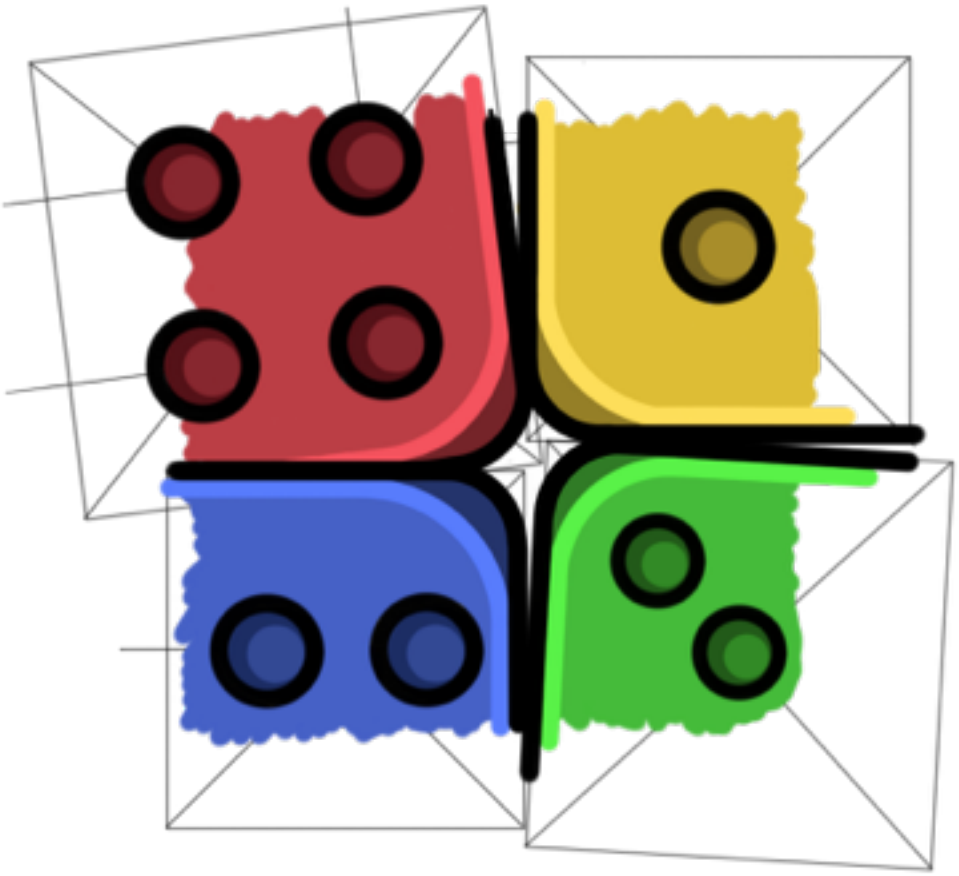
魔王骰子益智遊戲

實作 iOS 遊戲 App



逢甲大學資訊工程系四甲

指導：林明言 教授

學生：鄒獻忠、陳書恆

協力：馮廷瑋

2013年12月

摘要

本專題的研究內容包括： iOS 遊戲 App 專案如何規劃；使用 Trello 來做專案管理；cocos2D 的語法以及 Singleton 設計模式還有特有的 Delegate + Protocol（委託+協定）設計模式；使用 FMDB 的 SQLite API 實作於iOS系統上來完成遊戲對話模組；數學機率的方式來計算分數期望值、以及使用簡單的 and 、 or 、 not 、以及數學表示法來簡化程式碼；UX使用者體驗設計；資料結構則是使用簡單的Hash Table；以上這些元素是這份專題研究會呈現的內容。

這個專題研究特殊的地方有兩點：

一、對話模組的資料庫設計（我們稱之「感情模組」），這部份我們嘗試了SQLite的方式來儲存對話、角色表情圖片、動作圖片……等，。

二、使用者介面（UI）設計、以及使用者體驗（UX）設計將畫面的按鈕減低到最小，但是按鈕卻是隨使用者需求而出現的設計，讓玩家清楚知道該怎麼做，也降低學習的門檻。

我覺得這些是這份專題研究比較特殊、且值得研究實作的項目。

目錄

摘要.....2

1、導論.....4

1.1、專題概述與動機.....4

1.2、專題目標.....5

2、開發平台、iOS系統.....6

3、開發過程.....7

3.1、遊戲專案開始前的規劃.....7

3.2、初期的專案規劃、實作.....8

3.3、中期的專案規劃、實作.....10

3.4、後期的專案管理、實作.....12

4、技術使用.....14

4.0、簡介.....14

4.1、Singleton 單例.....15

4.2、Delegate 委託 + Protocol 協定.....18

4.3、cocos2d中 ccTouches 三個方法.....22

4.4、FMDB＋SQLite處理遊戲對話.....23

4.5、使用者介面的簡化、與提示.....27

4.6、使用簡單的機率統計計算分數期望值.....34

4.7、使用數學將程式簡化、提高效能.....36

4.8、遊戲計分動畫使用 Hash Table 資料結構.37

5、STD圖、遊戲畫面流程圖.....38

6、可用性測試 42

7、心得.....44

8、參考文獻.....45

9、附錄一：使用者測試報告.....46

圖二十六~三十四、遊戲畫面展示.....39~41

圖一、 遊戲畫面規劃與介面設計.....8

圖二、使用 iPhone 畫面大小來設計 UI 位置.....8

圖三、Logo以及學校的圖騰萌化.....9

圖四、Trello管理專案.....9

圖五、畫面切割與詳細規劃.....10

圖六、遊戲角色詳細設定之一.....11

圖七、遊戲角色詳細設定之二.....11

圖八、遊戲物件的圖案設計.....,11

圖九、遊戲資料結構設計、得分計算..12

圖十、詳細繪圖與著色.....13

圖十一、遊戲各類畫面規劃.....13

圖十二、Singleton 簡易解說.....15

圖十三、遊戲畫面物件、與飛機物件..... 18

圖十四、飛機物件發射飛彈.....19

圖十五、將 SQLite 加入專案要使用的 Framework 中.....25

圖十六、第一次玩的畫面.....27

圖十七、有存檔後玩的畫面.....27

圖十八、圖十九、使用者體驗設計範例 28

圖二十、二十一、使用者體驗設計範例 29

圖二十二、二十三、使用者體驗.....30

圖二十四、Database Schema.....31

圖二十五、遊戲畫面STD圖.....38

一、導論

1.1、專題概述與動機

2011年暑假，我在旁聽暑休的課程的時，跟同學討論到最近玩的一些創新的遊戲，這個遊戲融合了射擊遊戲（FPS）的節奏快的步調、以及傳統角色扮演（RPG）的等級、天賦系統，讓我廢寢忘食。當時剛開始盛行卡牌遊戲，於是我也開始構想卡牌遊戲結合傳統角色扮演遊戲的玩法：加入等級天賦，那麼遊戲一定會非常的有趣，但當時只是剛昇大二的學生，哪有能力去完成一個背後過於龐大個遊戲系統？於是我將目標轉小，先完成一款容易上手、簡單的小遊戲，讓自己技術、以及經驗提昇後，那麼才有可能完成更龐大、繁雜的卡牌遊戲，所以「魔王骰子」這個益智遊戲，就成了我專題研究的目標。

於是我開始學習 Objective-C，也就是 iOS 系統下的原生語言，並且開始看很多跟遊戲相關的遊戲設計的書籍，2012年初 Objective-C 原生語言大致上能了解且撰寫，也寫了幾款單一畫面、可以開啟來玩的小遊戲，但後來遇到的瓶頸是：遊戲架構在Open GL ES下，使用的是複雜的線性代數舉證轉換原理來處理圖像變化，這使的遊戲實作上出現無法突破的困難！

2012年暑假前，我買了「cocos2D遊戲程式開發供略」這本書籍， cocos2D簡單來說就是包裝Open GL ES的 Open Source API ，使遊戲製作有了突破性的發展，2012年暑假結束已經完成簡易版本的遊戲架構。

在製作期間為了未來軟體的可攜性，所以不少物件都有模組化，暑假後另一個夥伴陳書恆同學的加入帶來了SQLite的技術，並且使用了 FMDB 這個由外國人撰寫免費將 SQLite 包裝好的物件來實作遊戲角色對話以及事件；另外畫面人物存檔、讀檔使用 Singleton 設計模式；物件保持獨立的Delegate（委託）設計模式；資料結構部分，我們應用了 Hash Table，數學方面則使用一般機率論、簡單的 and 、 or 、 not 來簡化程式碼判斷，且在分數的計算上面讓期望值高於零，使玩家能充分體會遊戲的成就感、以及樂趣。

1.2、專題目標

完成一個完整的遊戲是最主要的目標，但是其中比較困難、且比較值得研究的部份是「對話模組資料庫化」，另外一個部分是設計方面，使用者介面設計、以及使用者體驗，讓玩家能在短時間內上手遊戲，並且融入其中，是我們這個專題的個別重要目標。

這裡我們將目標更清楚的列出來：

一、專案管理方式（使用Trello）

二、遊戲軟體的專案規劃

三、Objective-C物件導向設計及設計模式

四、cocos2D for iPhone 的運用

五、對話模組資料庫化（SQLite以及FMDB）

六、UX使用者體驗設計

二、開發平台、iOS系統

2.1、開發環境

Xcode 是蘋果電腦為開發員提供的整合式開發環境。The Xcode suite 包含有 GNU Compiler Collection，支援 C語言、C++、Fortran、Objective-C、Objective-C++、Java、AppleScript、Python 以及 Ruby。

Xcode 為 Apple 內建的免費編輯器，提供了代碼補全、語法高亮、代碼摺疊，還有內置註釋的錯誤、警告和說明。Xcode 中的測試導航器可以簡單的創建、編輯和運行單元測試，使得測試驅動的開發更簡單。

2.2.1、軟體框架

採用了 cocos2D for iPhone 做為開發框架。cocos2D 是一個開源的2D遊戲框架，其特點是該 API 集成了 OpenGL （Open Graphics Library，開放圖形庫），但使用 cocos2D 系列的引擎無須掌握 OpenGL 的相關知識，從而降低開發難度。

2.2.1、開發語言

Objective-C 是一種通用、高階、物件導向的程式語言。它擴充功能了標準的ANSI C程式語言，將Smalltalk式的訊息傳遞機制加入到ANSI C中。

SQLite 被封裝成一個庫（單一個檔案），因此具有相對小的體積，經由直接在程式語言內的 API 呼叫，對於整體設計上具有減少消耗和延遲的作用。

2.3、iOS系統

iOS是由蘋果公司為行動裝置所開發的操作系統，其特點是Unix-Like，且核心底層處理了多點式觸控，讓觸碰的應用發揮到淋漓盡致；其支援的裝置包括iPhone、iPod touch、iPad……等；iOS不支援非蘋果的硬體。

三、開發過程

3.1、遊戲專案開始前的規劃

在開始動手寫遊戲專案之前，如果有一個自己認為好玩的遊戲構想，是不會馬上坐到電腦前面，然後開始撰寫文件、做美術設計、或是動手寫程式碼，我的作法是：用手邊的實體工具來測試所想的遊戲內容是否有遊戲性（遊戲是否可吸引人去玩）。

Demon Dice （魔王骰子）這遊戲是利用機率的方式來隨機產生數字 1 ~ 4 的骰子，而每個骰子又有四種顏色（紅、黃、藍、綠），這些產生出來的骰子必須擺在一個 4 x 4 的棋盤上面，每次擺放完畢就會計算分數，但是骰子固定了以後是無法刪除的，要刪除必須符合遊戲的四種規則：

一、顏色全相同、且數字也相同的四個骰子排成一列。

二、顏色全相同、且數字全相異的四個骰子排成一列。

三、顏色全相異、且數字全相同的四個骰子排成一列。

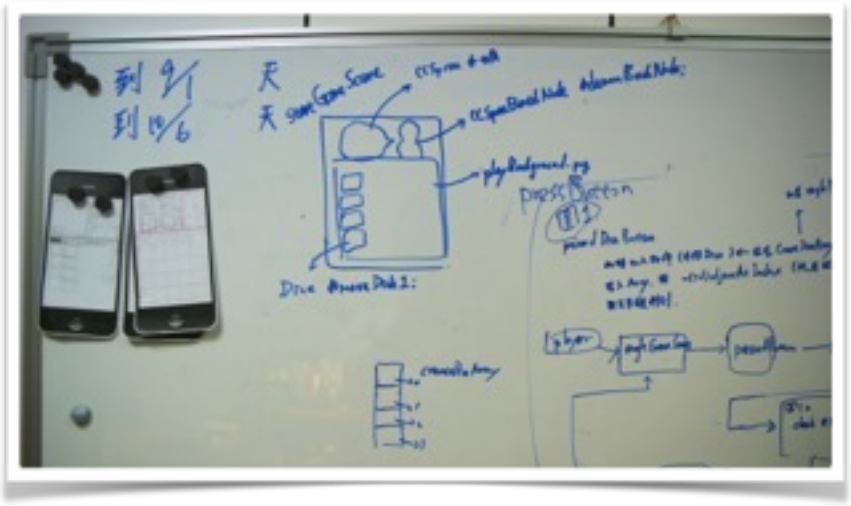
四、顏色全相異、且數字全相異的四個骰子排成一列。

既然有了遊戲的構想，於是我去商店買了 16 顆骰子，然後直接使用手動的方式骰骰子、利用這個遊戲規則來試玩看看，經過測試後覺得這個遊戲的遊戲性是可以吸引人的，這時候才決定開始著手專案的規劃。

當時我們製作團隊很快的就決定使用 cocos2D for iPhone 這個Open Source 的遊戲 API ，原因是這個 API 將 Open GL ES 處理圖像的引擎包裝起來，避免使用繁雜的線性代數來增加遊戲實作上的難度，如果要我比喻，我會將「製作遊戲」比喻成「蓋一棟大樓」，而使用 Open GL ES 猶如從製作磚塊開始蓋起，使用 cocos2D 就像是模組化且創建好的牆壁、房間等等的工具供您使用，可以省略很多跟創作遊戲無關的瑣碎細節，而開發者只要想著遊戲的實現以及困難的解決就好。

3.2、初步的專案規劃、實作

確定遊戲玩法、以及內容以後，就可以開始著手處理專案的規劃了，團隊們認為遊戲 App 跟一般 App 有很大的不同點：也就是畫面的呈現。因此在規劃一個遊戲的時候，以畫面為主要方式來製作遊戲的流程，也就是說，直接用圖像來表達遊戲畫面的呈現、以及畫面的按鈕按了以後，會有什麼樣的反應、或是該跳到哪個畫面等等。



圖一、 遊戲畫面規劃與介面設計

當時影印了不少 iPhone 大小相似的實際圖案，並且在上面直接繪製畫面呈現的樣子、以及哪裡該有按鈕、按下去後該轉到哪個畫面，並且這時候就可以把畫面的 UI（使用者介面）的元件大小、以及元件的位置計算出來，然後開始著手美工設計了。

圖二、使用 **iPhone** 畫面大小來設計 **UI** 位置

接下來會開始規劃遊戲內該有的物件、以及物件內該有的屬性、變數命名規則，決定以後會先思考每個物件之間的關聯性、是否可以簡化、或是繼承等等，（這部份我們會在下一章節詳細說明，這裡先不說細節。）確定物件以後，就能開始動手寫點程式碼了。

圖三、**Logo**以及學校的圖騰萌化

為了能夠更有條理的管理專案，團隊使用了 Trello 這個專案管理軟體，這個軟體是免費的，且所有瀏覽器都可以開啟並管理。團員們會把該做的工作丟進 ToDo 的 List 之中，每次開始寫專案的時候會先去 Trello 中，查看還有什麼事情該做，開始實作的時候就把事情丟入 Doing 的 List，而有什麼要注意的地方也可以馬上在該做的事情下方註明，完成的話就會丟到 Done 的 List 留存，底下有幾張我使用的照片。

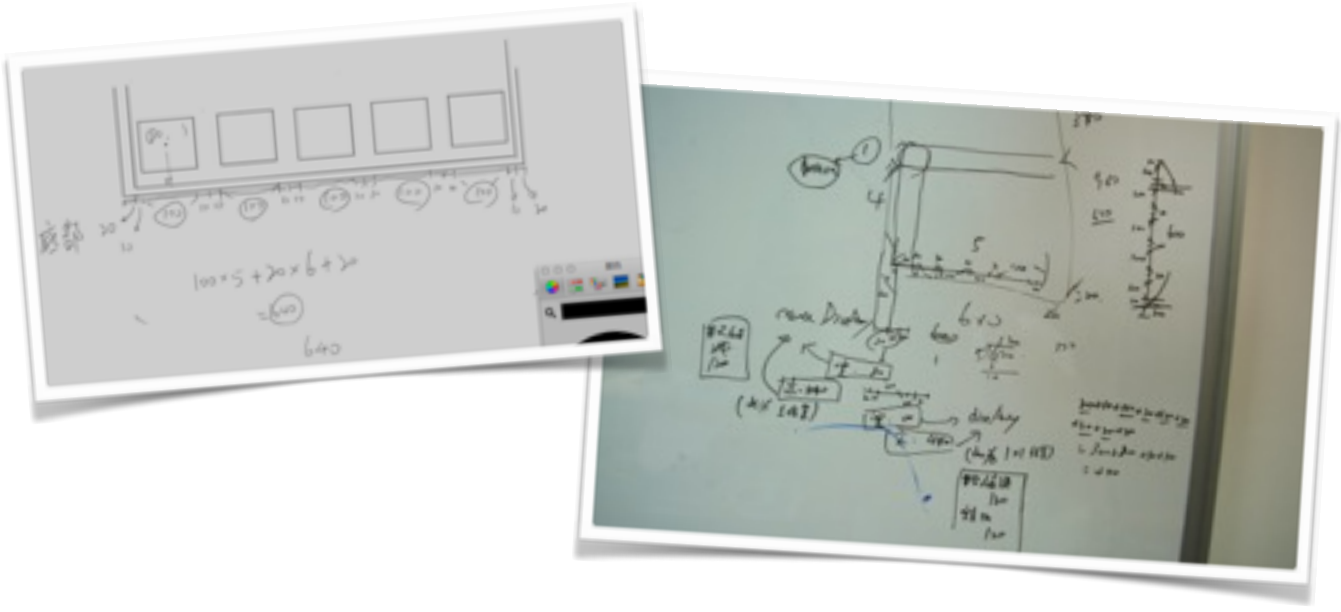
圖四、**Trello**管理專案

3.3、中期的專案規劃、實作

這個時期是時間最長的，由於團隊採用的方式是「快捷開發法」，所以每開發到某個進度，就會確認自己是否有達到需求，比方說：進入遊戲後一開始的選單畫面，會先實作完成後，確認是否有跑正確的選單動畫，讓玩家知道哪裡是按鈕，並暗示使用者該如何做下個動作……等，完成後我會開始實際玩看看並且測試，接著除錯。

而這些流程會不斷的不斷的重複著：修正設計、實作、確認完成內容、測試且除錯，一直到確認完成此畫面的所有需求。這中間我們會開始繪製流程圖、以及遊戲的細節，如計算分數的方式、或是該用哪種資料結構來實踐、使用巨集及 typedef enum（列舉型態）讓程式碼可讀性更高……等，讓程式更清晰易懂，並且會把功能做切割，避免除錯不易、以及要加入新功能的時候，會因為函式與函式之間、物件與物件之間綁的過於緊密，導致複雜度升高（這部份真的很重要！）這些都是這個時期必須注意的事情，並且在實作的同時也要思考如何切割，因為有時候必須做了以後才會發現問題所在，那時候在來做更詳細的切割會比遲遲卡在第一階段想太多，而不去實作來的有效率。（這部份細節會在下一章說明。）

另外在這個階段會做更詳細的遊戲角色設定，讓角色更鮮明，以及畫面更詳細的切割讓使用者介面看起來更美觀、符合單手拿著把玩的需求。



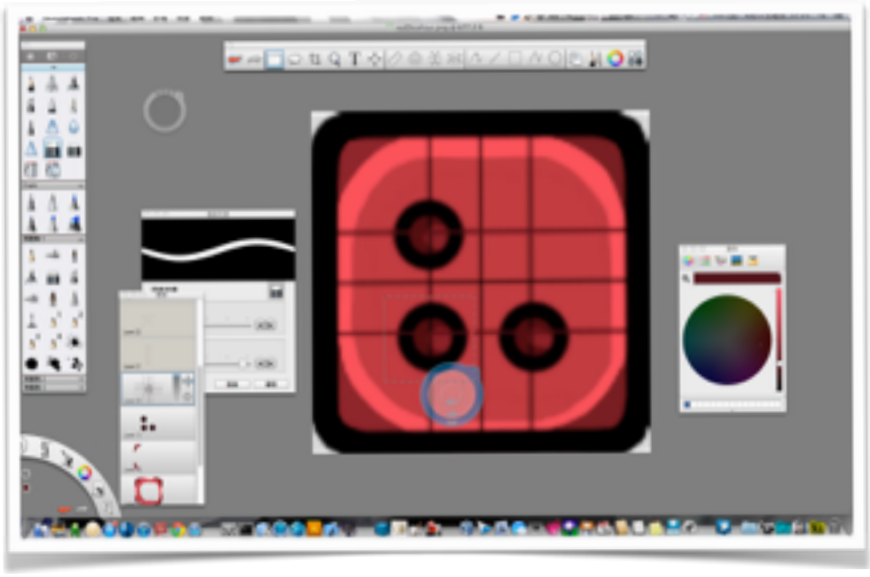
圖五、畫面切割與詳細規劃

圖六、

遊戲角色詳細設定之一

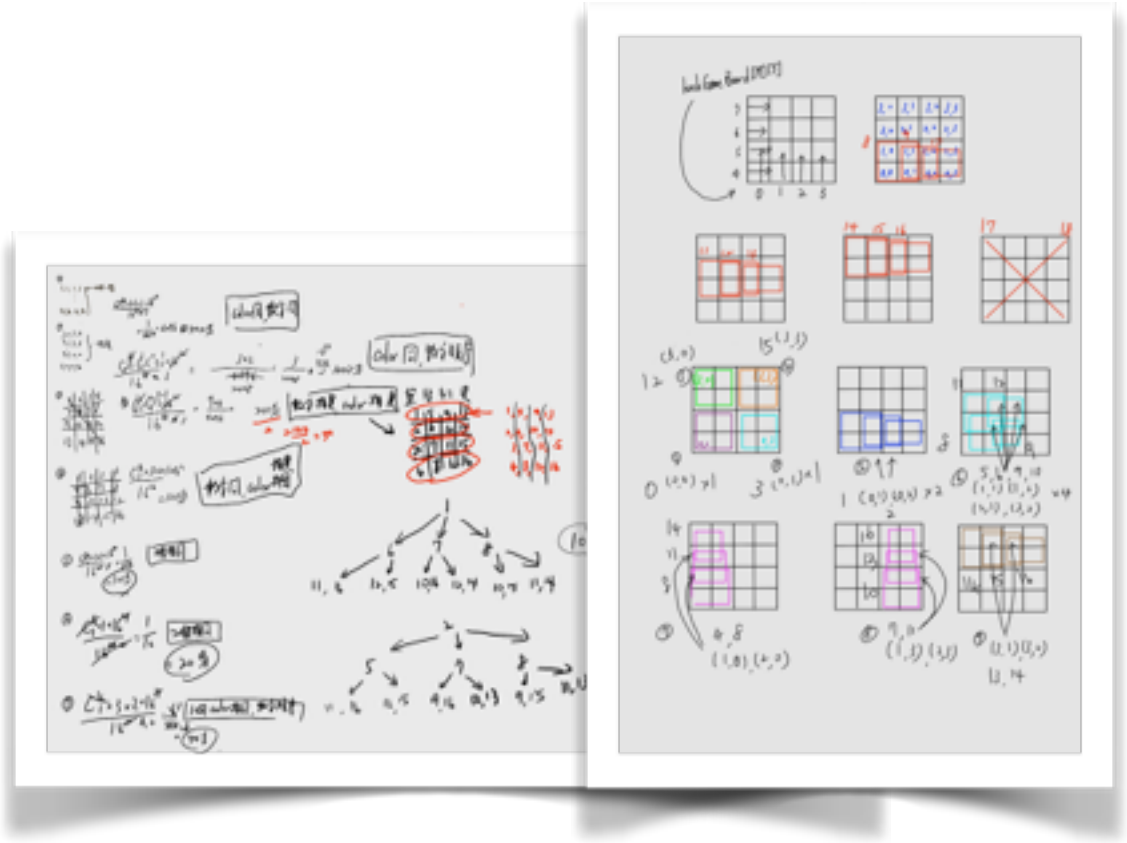
圖七、

遊戲角色詳細設定之二



圖八、

遊戲物件的圖案設計



圖九、遊戲資料結構設計、得分計算

3.4、後期的專案管理、實作

當遊戲完整度越來越高的時候，建議這時候可以檢視自己過去完成的部份，並且查看是否可以在不改變遊戲邏輯的情況下，將程式碼做重構的動作，這裡所謂的重構是視情況而去做的，比方說：當時遊戲的分數判斷邏輯寫的很複雜，原因是使用人腦的直觀判斷方法有時候並不是用於程式的邏輯判斷，因此我們花了時間將這部分邏輯判斷寫成數學的代數表達式（比方說 X + Y + Z + W = 34），來代表每個骰子計分時的總和，然後再來簡化邏輯判斷的流程以及找出決定性的代數表達式，這時候再轉變成程式碼，這個動作讓我們從將近 200 行的程式碼，縮短到 50 行左右，且由於判斷次數減少，效能也提高了，以每秒 30 偵的速度來算，如果以減少一次的邏輯判斷來計算，那麼每秒至少減少 30 次的判斷，且程式碼也更精簡易懂。

當程式部分開始做總結的時候，繪圖與視覺上的設計會是這個階段的重點，另一方面會找很多沒玩過遊戲的玩家來直接測試，並且在旁記錄缺失與改進項目，寫進 Trello 專案中來處理。

圖十、詳細繪圖與著色

圖十一、遊戲各類畫面規劃

四、技術使用

4.0、簡介

這裡先介紹一下在遊戲中會用到的技術，為何使用？如何用？如何延伸使用？先約略說明一下我在遊戲中用到的技術層面：

一、Singleton （單例）

二、Delegate （委託） + Protocol （協定）

三、cocos2D 中的 ccTouches 三個方法

四、FMDB + SQLite 資料庫取代傳統 XML 對話的方式

五、使用者介面設計的簡化、與提示

六、使用簡單的機率統計方式來計算分數期望值

七、使用數學將程式簡化、提高效能

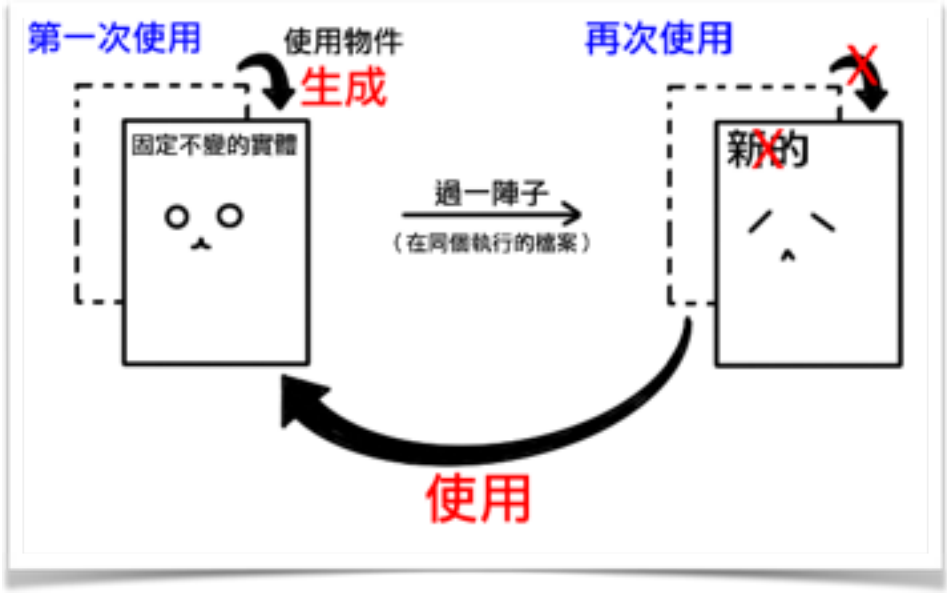
八、遊戲計分動畫使用 Hash Table 資料結構

4.1、 Singleton 單例

簡單說明一下單例是什麼。

單例是一種物件導向的設計模式，幾乎所有物件導向都有這種形式的設計模式，這個設計模式特別的地方是：只要是單例的物件，在被創造出來以後（產生實體），那麼就只會存在唯一的一個實體物件，也就是當其他人要使用、甚至又想創造新實體的時候，是無法去創造另一個新的實體來使用，而是會一直使用最初的實體物件。

說起來很抽象，我用圖示來表達：



圖十二、**Singleton** 簡易解說

概念就如同圖示，只有第一次會生成物件的實體，接著後面要使用的時候都會用同樣的實體，而不會去產生新的實體。

這個特性有很多好處，如：遊戲畫面的切換，畫面一定要每次都只有一個實體在進行，否則很容易混亂、甚至會造成記憶體不足而當機；玩遊戲的時候，玩家的存讀檔系統也必須用 Singleton 設計模式，玩家在玩遊戲通常都會是一個人在遊戲裡玩，在裡面切換畫面，不會因為畫面切換而每次生成新的玩家物件，然後載入物件，這樣做不合理也非常消耗效能與記憶體空間，更容易因為切換錯誤造成玩家檔案錯誤；另外

還有資料庫也是用 Singleton 來實作，才不會造成不曉得哪個生成的物件去存取了資料庫，更改資料庫、或是開了資料庫沒有關閉，而造成記憶體遺漏……等。

這裡寫一些 Singleton 比較重要的內容，讓我們能更了解在 Objective - C中該如何實作 Singleton 設計模式（這裡我舉如何用 Singleton 實作處理遊戲畫面為例子，物件名稱為 GameManager ）：

```
/* GameManager.h 檔案部分內容 */
@interface GameManager : NSObject {
        EnumSceneTypes currentScene;
  }
+(GameManager *)sharedGameManager;
-(void)runSceneWithID:(EnumSceneTypes)sceneID;
@end
```

物件裡只有一個 currentScene（目前畫面）這個類別成員，而前面的屬性是我使用 Enum 列舉型態去自定的。

另外要注意的就是下面的 +(GameManager *)sharedGameManager; 這個類別方法，類別方法就是不用產生實體也可以使用的方法，所以當我要呼叫更換畫面的時後，就會用這個方法，這個方法會回傳「固定不變的」實體給我。

而 -(void)runSceneWithID:(EnumSceneTypes)sceneID; 是當我要切換畫面時候，會用的實體方法，我會去找出現在要換成哪個畫面然後更換。使用的寫法如下：

```
[ [GameManager sharedGameManager] runSceneWithID: 畫面ID ];
```

前面 [GameManager sharedGameManager] 會回傳一個「固定不變的」物件，然後這個「固定不變的」物件，會用 runSceneWithID: 來更改目前畫面，達成遊戲畫面切換的效果！

```
/* GameManager.m 檔案部分內容 */
@implementation GameManager
static GameManager *_sharedGameManager = nil;
```

這裡的 static 固定變數就是用來存「固定不變的」實體的記憶體（所以他是指標）

```
+(GameManager *)sharedGameManager{
        @synchronized([GameManager class]){
                if( !_sharedGameManager ){
                        [[self alloc]init];
                }
                return _sharedGameManager;
        }
        return nil;
}
```

@synthesized 這個是 Objective - C 為了讓多線程單一化而特有的方法，用來讓這個物件即使多線程也不會同時去創造這個物件而造成問題。

```
+(id)alloc{
        @synchronized([GameManager class]){
                NSAssert(_sharedGameManager == nil , @"Attempted to allocate a second instance of the Game Manager singleton");
                _sharedGameManager = [super alloc];
                return _sharedGameManager;
        }
        return nil;
}
```

alloc 是生成物件實體時必須使用的方法，生成實體部分會確認如果是第一次生成，那就真的生成實體；如果已經有實體，那麼就直接回傳「固定不變的」實體就好。

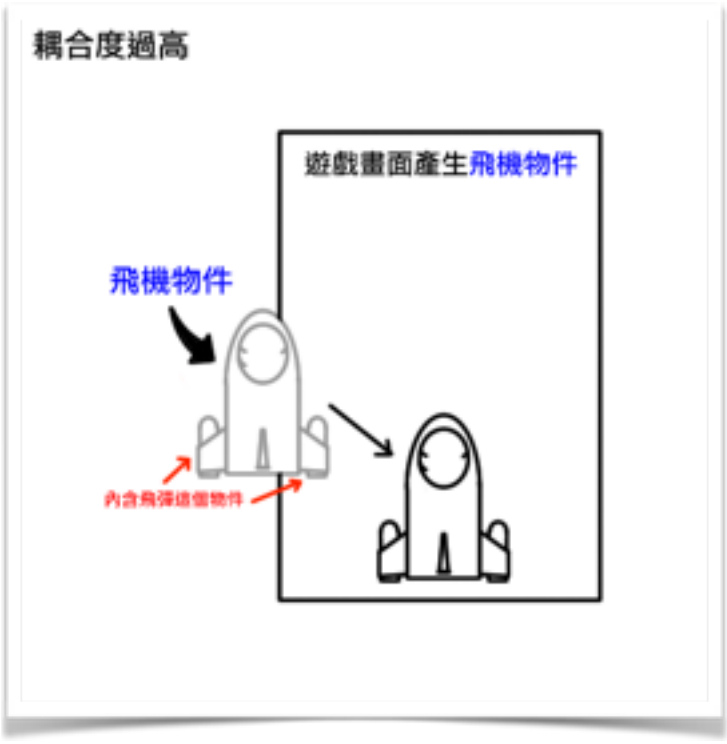
4.2、Delegate （委託） + Protocol （協定）

在Objective - C 這個物件導向語言中，委託被使用的淋漓盡致，原因是因為他有個資料型態，叫做 id ，這個資料型態能接受任何「只要是物件」的形態，因此在物件裡面連「自己」都可以回傳給其他物件來使用，使的物件之間傳遞變得非常靈活。

Protocol 協定在 Objective - C 中，是用來將「方法」獨立出來，使「不同的物件」可以使用「相同的方法」的一種設計模式，比方說：當你做了一個小畫家軟體，鉛筆這個物件可以畫出「線」，但是水彩筆也可以畫出「線」，這時候方法名稱為了讓他統一以方便程式呼叫使用，於是就把方法包裝成協定，讓「畫」這個方法可以在動態聯繫的時候，即使不知道到底這時候是鉛筆要畫、還是水彩比要畫，但是都可以做畫的動作。

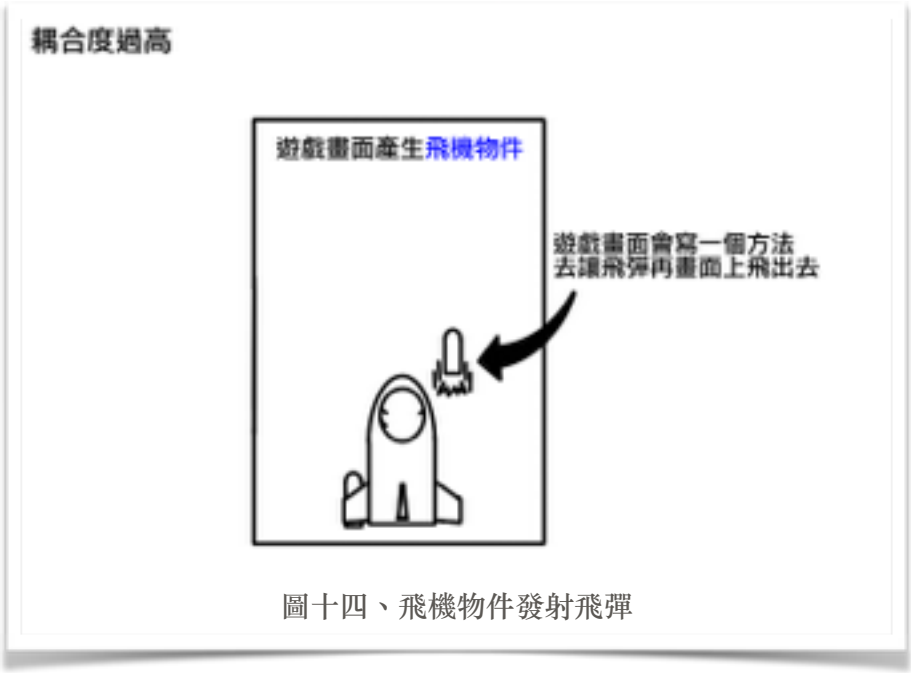
Delegate 委託 + Protocol 協定是 Objective - C 中為了防止物件與物件之間耦合度太高，導致所建立的物件被綁死在某個程式之中，無法在被利用，而這違背了物件導向所謂的「物件再利用」的想法，我們先來看看物件耦合度過高的情形。

如右圖，在物件導向中，畫面也是一個物件，而這個物件因為要包含飛機，所以我們就會在這個畫面生成飛機這個物件。



圖十三、遊戲畫面物件與飛機物件

當玩家按下發射飛彈按鈕的時候，遊戲場景會出現飛彈物件發射出去，那是因為我們在遊戲場景上寫了一個方法，讓飛機物件裡面的砲彈物件飛出去。



圖十四、飛機物件發射飛彈

問題來了：假設這個場景必須重複使用，但是未來這個遊戲場景裡面沒有太空船，那麼這個場景物件就必須要把「飛彈飛出去」有關的方法全部找出來刪除，更不用說當你創造物件的時候你會知道要去寫「飛彈飛出去」這個方法嗎？這樣子就違背了物件導向的「物件重複利用」的初衷，因此 Objective -C 這個物件導向就有了 Delegate + Protocol 來解決物件與物件耦合度過高的問題。

他們是如何解決的呢？用剛剛飛機、飛彈以及場景物件的例子先說明一下原理：

一、 飛機物件裡面多了一個可以接收任何物件型態的變數，我們通常都會把這個變數名稱定為 delegate，以下是飛機物件 Airplane.h 的寫法：

```
@interface Airplane : CCSprite {
        id  <AirplaneDelegate>  delegate ;
}
@property (nonatomic , weak) id <AirplaneDelegate> delegate ;
@end
```

id 就是前面提過的 Objective - C 內可以接收任何物件型態的萬用型態，中間的 < AirplaneDelegate > 的意思其實是說這個delegate 是由 AirplaneDelegate.h 而使用的，讓程式設計師知道是哪個協定要使用的而不會混亂；所以就是說當你（畫面）生成這個

Airplane 物件時，你（畫面）本身就有責任要去實做這個協定，到時候記得要把協定的內容補齊才能正常使用其方法。最後變數名稱就設定為 delegate 比較好辨識，後面 @property … 是讓這個變數可以外部存取。

```
@implementation  Airplane
@synthesize delegate;
//初始化讓 delegate = nil;
-(void)fireBomb{
        [delegate  fireBombDelegate];
}
```

這裡重點在於下面那個方法，裡面看起來好像是呼叫一個方法就結束了，但是變數的 delegate 初始化的時候是空的，這樣不會有問題嗎？而那個 [delegate fireBombDelegate]; 中的 fireBombDelegate 是從哪來的？其實就是 AirplaneDelegate 中的方法來的，下面是協定 AirplaneDelegate.h 的程式碼：

```
@protocol AirplaneDelegate <NSObject>
-(void)fireBombDelegate;
@end
```

就那麼簡單，只要寫出一個名稱就好，因為他只是讓其他物件如果有跟這個協定定下協定以後，你就要去實作這個方法，這個方法是寫在定下協定的物件裡面，不是寫在協定本身。底下是物件該做的，分成 GamePlayeLayer.h 以及 GamePlayeLayer.m 檔案：

```
// GamePlayLayer.h
@interface  GamePlayLayer : CCLayer <AirplaneDelegate>{
}
```

在個物件名稱最後面（也就是GamePlayLayer 最後面的誇號），寫上 <AirplaneDelegate> 就表示我這個物件和這個協定定下關係了，因此我必須在 .m 檔案之中實作 -(void)fireBombDelegate; ，最後還有兩步很重要的步驟才能完成 Delegate + Protocol。

```
// GamePlayLayer.m
@implement  GamePlayLayer
-(id) init {
        self = [super init];
        if( self != nil ){
                Airplane *testAirplane = [[Airplane alloc]init];        // 生成飛機物件
                [testAirplane setDelegate: self ];     // 這句就是完成Delegate的第一個重點！
        }
}
-( void )fireBombDelegate{
        /* 請來這實作發射飛彈！ */
}
@end
```

當你在遊戲畫面生成這個飛機物件的時候，你會把畫面本身自己的物件傳送給 Airplane 中的 delegate ，因此回頭看之前飛機物件中的：

```
-(void)fireBomb{
        [delegate  fireBombInScene];
}
```

就很合理了，不是嗎？

使用這個方法就可以不用擔心兩個物件耦合度過高，因為當畫面要單獨使用的時候，不會有一個方法被飛機中的炸彈飛行動作綁死；另一方面，當我要把飛機拿到其他畫面也很容易實作，只要那個畫面生成飛機的時候，記得要去實作要求的協定的方法，然後畫面生成飛機的時候記得把自己本身委託給飛機的 delegate 中，讓飛機受到畫面委託去做事情（但是其實還是必須要在畫面內把委託的方法實作才能正常發射飛彈），於是就解決物件耦合度過高的問題了！

4.3、cocos2D 中的 ccTouches 的三個方法

在 iOS 系統中，最重要的就是觸碰的處理，所以在 cocoaTouch 之中，也有四個方法可以處理觸碰，但是 cocos2D 將這四個 cocoaTouch 的 Delegate 拿來加以包裝，讓觸碰更容易使用！底下是這三個經過 cocos2D 包裝後的方法：

```
-(void)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
        /* 開始觸碰的那一瞬間 */
}
-(void)ccTouchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
        /* 接續前面沒有中斷時，會一直執行這個方法（看每秒偵數） */
}
-(void)ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
        /* 離開畫面的那一剎那，會執行這個方法 */
}
```

使用這三個方法，就可以看遊戲需求做出各種動作，比較值得一講的是，這些方法中所接收的 (NSSet *) touches 、以及 (UIEvent *) event 這兩個函式接收值。

先說明一下這兩個類別： NSSet 以及 UIEvent ，NSSet 是一個集合，這個集合內容當被創造了以後內容是不可變更的，且是每個都是唯一不重複的，就跟數學的集合概念一模一樣，在這裡，他會收集觸碰螢幕上你所碰到的點，存入這個集合中。而 UIEvent 會接收一個 iOS 所擁有的事件，在 iOS 系統中，有三種 Event 事件：

一、Touch Event （觸碰事件）

二、Motion Event （動作事件，如：搖動手機等等）

三、Remote Event （遙控的事件，如：用手機操控投影片等等）

這裡我們當然是用 Touch Event ，不過由於單點觸碰是不用使用到所接收的 UIEvent ，多點觸碰才會用到，所以在這裡暫不討論。

而我們要怎麼使用這三個觸碰方法所接收的 (NSSet *) touches ？這裡簡短的介紹如何使用接收值來處理「單點觸碰」的方法：

```
UITouch *myTouch = [touches anyObject];
CGPoint  touchLocation = [myTouch locationInView:[myTouch view]];
touchLocation = [[CCDirector sharedDirector]convertToGL:touchLocation];
```

第一行是將所接收的 NSSet 轉成 UITouch ，第二行是將剛剛的 UITouch 轉成座標點，重點是最後一行，這裡要特別說明。

在 iOS 系統下，他的座標點是以左上角為圓點， x 軸往右增加， y 軸往下也是增加，但是在 cocos2D 是使用正常的數學座標， 因此必須要使用 convertToGL : 這個方法，將觸碰到的座標點轉換成 cocos2D 中 GL 的座標系統。

4.4、FMDB ＋SQLite 資料庫來處理遊戲對話

一般遊戲的角色對話，通常都是用 XML 固定存於文件當中，要使用再一一載入對話來實現。在專題中，為了讓遊戲人物的對話可以增加其可攜性、未來擴充性、以及將來遊戲內 NPC 可以隨著玩家喜好而改變個性以及對話內容，讓遊戲能更耐玩且吸引人。

手機上的資料庫一般不會處理到伺服器等級的幾百萬次存取，因此我們選用 SQLite，使用SQLite有很多好處：免費、支援大多數的 SQL 指令、一個檔案就是一個資料庫，不用安裝資料庫伺服器軟體、完整支援 Unicode、速度快，但是如果要用原生 Objective - C 來實現還是有一定的難度，因此我們採用了開源碼 FMDB 來處理 SQLite， FMDB 是由 ccgus 這個程式設計師所分享的一個把 C API 包裝成簡單易用的 Objective-C 物件，讓我們能夠使用字串就能輕鬆存取資料庫內對話內容，再顯示到遊戲的對話視窗上面！實作的方式大致說明一下：

一、實作一個 Singleton 物件專門處理資料庫

二、將 FMDB 的 API 匯入專案中

三、再專案中將 SQLite 的 Framework 加入專案

四、寫讀取對話資料的指令及相關程式

先說明實作一， Singleton 物件部分就不提了，比較值得注意的是由於專案再跑的時候並不是再專案檔案夾中執行， Apple 公司為了保護安全會設定一個資料夾來存放檔案、以及測試下載的檔案是否有符合規定，所以我們在這個處理資料庫的 Singleton 物件中，必須加入一個方法將我們建立好的檔案加入 Apple 規定的資料夾中，底下是實作方法：

```
-(NSString *)getDBPath{
        NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);      //Documents
        NSString *documentsDir = [paths objectAtIndex:0];
        return [documentsDir stringByAppendingPathComponent:@“test.sqlite"];
}
```

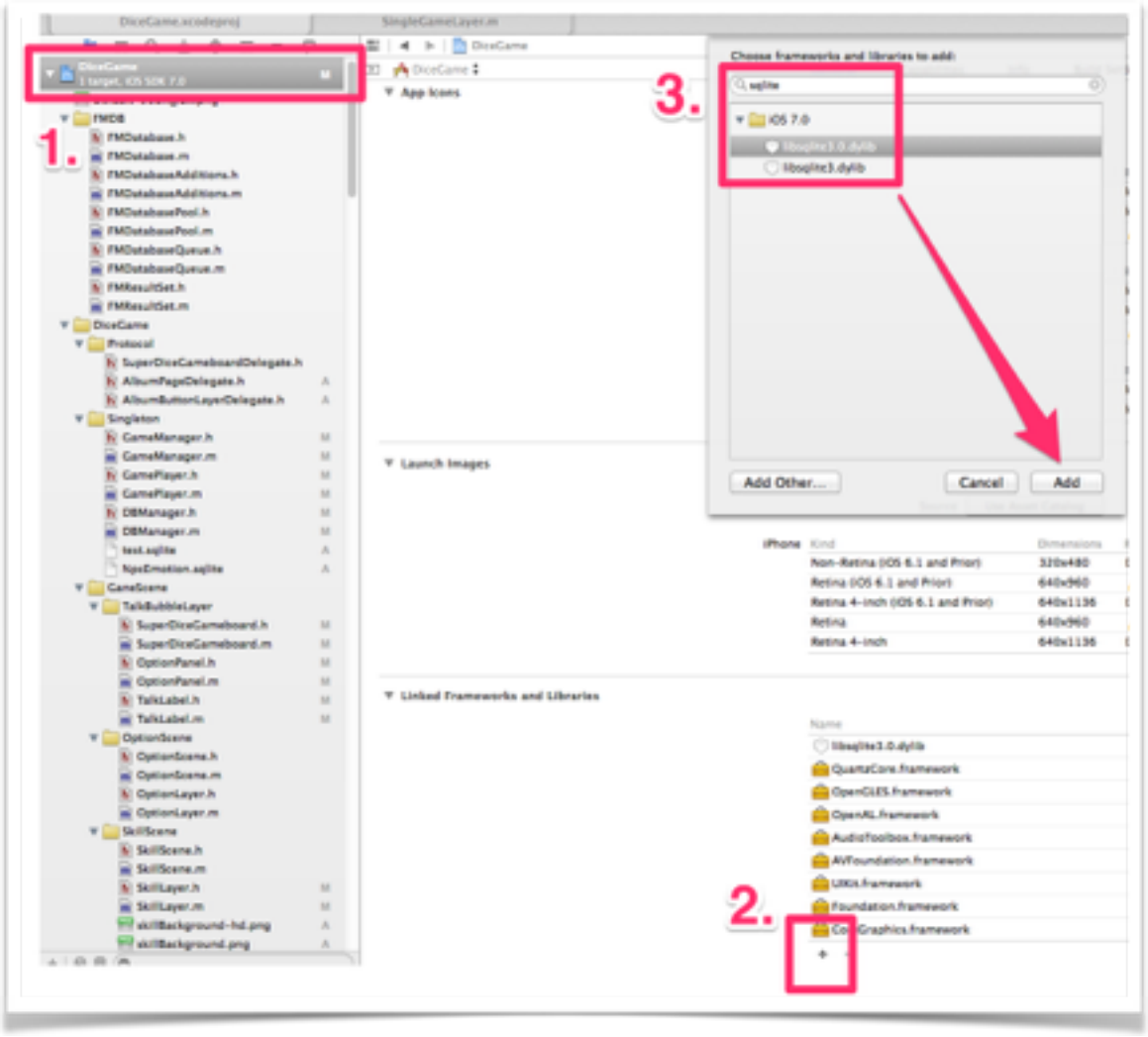
上面這個方法是將「專案夾的路徑」給找出來。

```
-(void)CopyDatabaseIfNeeded{
        CCLOG(@"將 .sqlite 複製到專案上面！");
        NSFileManager *fileManager = [NSFileManager defaultManager];
        NSError *error;
        NSString *dbPath = [self getDBPath];
        BOOL success = [fileManager fileExistsAtPath:dbPath];
        if(!success) {
                NSString *defaultDBPath = [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"test.sqlite"];
    CCLOG(@"複製路徑：\n%@",defaultDBPath);
                success = [fileManager copyItemAtPath:defaultDBPath toPath:dbPath error:&error];
                if (!success){
                        NSAssert1(0, @"Failed to create writable database file with message '%@'.", [error
localizedDescription]);
                }
        }
}
```

這部分是將專案複製到 Apple 規定存放檔案的地方，如果失敗會回傳訊息。

實作二的部份只是將 ccgus 所寫的 FMDB 物件拖拉複製近專案中。

實作三則是看下圖就可以了解：



圖十五、將 SQLite 加入專案要使用的 Framework 中

實作四，語法跟 SQL 的寫法是一樣的，比較要注意的就是 FMDB的用法：

```
-(id)queryContext{
        NSString *tempContext;
        [self loadDB];
        NSString *selectString = @"SELECT Context FROM Dialog";
        FMResultSet *rs = [gamePlayDB executeQuery:selectString,randomNumberIDString];
        while([rs next]){
                tempContext = [rs stringForColumn:@"Context"];
                NSLog(@"context = %@" , tempContext);
        }
        [rs close];
        [gamePlayDB close];
        return tempContext;
}
```

要使用的時候必須先使用 loadDB 來開啟資料庫，使用完要記得使用 close 來關閉資料庫，另外在 FMResultSet 這個 FMDB 的物件中，不管讀到幾個物件，一定要用 while 迴圈去一個個讀出資料，否則是抓不到東西的。

PS：Database Schema 、以及資料庫說明在第31 ~ 33頁。

4.5、使用者介面設計的簡化、與提示

在玩新入手的遊戲時，會讓玩家不自覺而持續玩下去的其中一個關鍵在於：遊戲上手的程度。因此在設計流程的時候，我們設計了很多雖然看不見，但是隱藏在內的設計，底下有幾個圖示來表達：

圖十六、第一次玩沒有存檔時，

不會顯示刪除檔案的按鈕

圖十七、有存檔時，

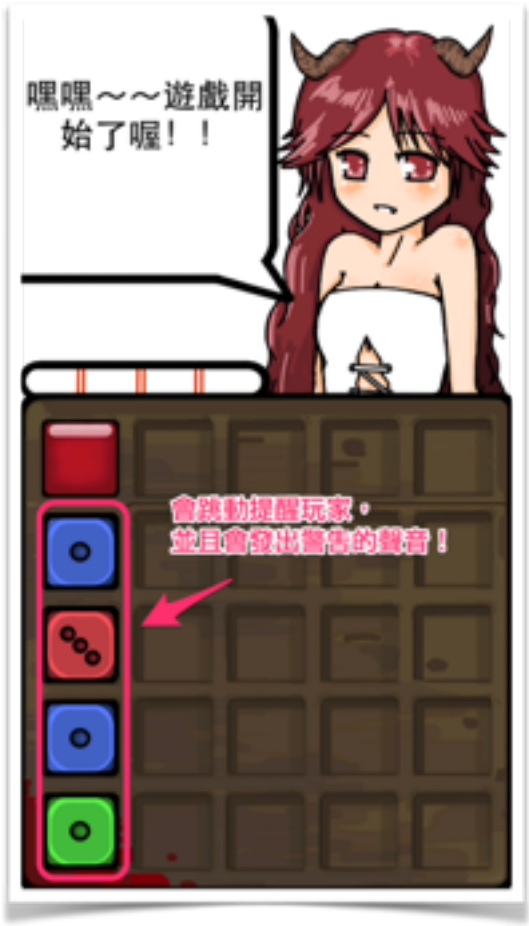
會顯示刪除檔案的按鈕

玩家在玩遊戲的時候不該出現的按鈕就不會出現，，讓玩家能專注在遊戲上，按鈕數量盡量少於 4，且所有按鈕都會有「出現彈跳出」的簡易動畫，提醒玩家「這裡可以按」，並且加上按下去的陰影，讓玩家有按下立體按扭的錯覺。

圖十八、如果沒有檔案，會自動進入新手教學

圖十九、有檔案，則進入遊戲

（新手教學會再?號圖示中重複查詢觀看）



圖二十、玩家再玩遊戲的時候，如果還有骰子沒有移動，會有該移動骰子跳動的動畫來提示玩家應該把骰子移動完。

圖二十一、棋盤擺滿以後，假設還有骰子可以移動，畫面會直接顯示還可以移動的骰子提醒玩家！

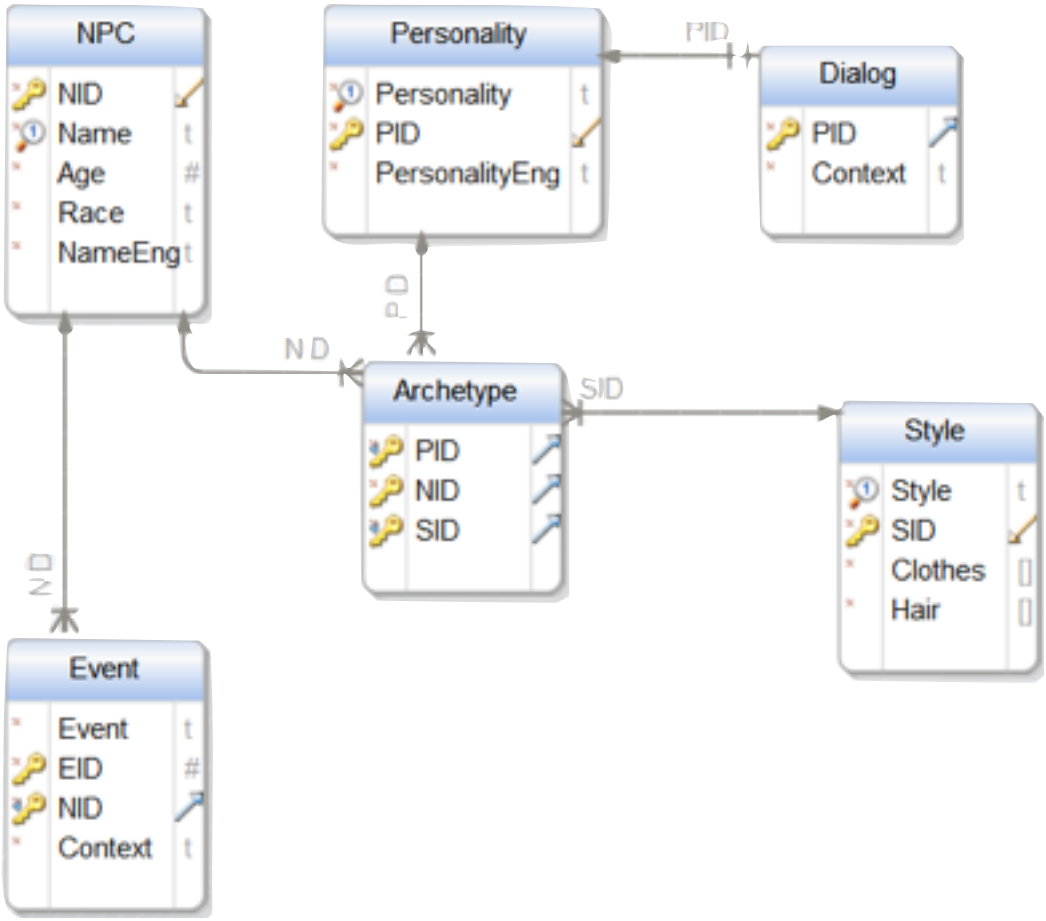
X

圖二十二、原本設計：按鈕在上，顯示文字在下。但是如果以拿手機的姿勢這樣會擋到文字，且下方字體容易被擋住。

O

圖二十三、更改設計：按鈕在下，顯示文字在上。以使用者拿手機玩遊戲的姿勢較為人性化且合理。

附件一、Database Schema



圖二十四、

Database Schema

附件二、資料庫說明

Archetype 原型

欄位	意義	中文	性質	資料型態
NID	NPC ID	角色代碼	外鍵	整數
PID	Personality ID	個性代碼	外鍵	整數
SID	Style ID	造型代碼	外鍵	整數

角色原型定義，因為正規化NPC、PERSONALITY和STYLE而產生的新表。

NPC 非玩家控制角色

欄位	意義	中文	性質	資料型態
NID	NPC ID	角色代碼	主鍵	整數
Name	角色名稱	中文名稱	唯一	字串
Age	年齡	同左		數字
Race	種族	同左		字串
NameEng	角色名稱	英文名稱		字串

NPC的基礎設定，不會變的設定。

Event 事件

欄位	意義	中文	性質	資料型態
EID	Event ID	事件代碼	主鍵	整數
NID	NPC ID	角色代碼	外鍵	整數
Event	事件	事件	不可為空	字串
Description	事件內文	事件描述		字串

遊戲內特殊事件

Personality 個性

欄位	意義	中文	性質	資料型態
PID	Personality ID	個性代碼	主鍵	整數
Personality	個性	中文個性	唯一	字串
PersonalityJap	個性	日文個性		字串
PersonalityEng	個性	英文個性		字串
Description	個性描述	同左		字串

性格/人格特質設定

Dialog 對話

欄位	意義	中文	性質	資料型態
DID	Dialog ID	對話代碼	主鍵	整數
PID	Personality ID	個性代碼	外鍵	整數
Context	對話內文	同左		字串

性格具有的對話資料庫

Style 造型

欄位	意義	中文	性質	資料型態
SID	Style ID	造型代碼	主鍵	
Style	造型	同左	唯一	
Description	造型描述	同左		
Hair	髮型	髮型圖片		圖片
Face	臉部	臉部圖片		圖片
Clothes	衣裝	衣裝圖片		圖片

NPC複雜設定，會因事件、劇情改變

4.6、使用簡單的機率統計計算分數期望值

由於我們遊戲玩法是：

一、顏色全相同、且數字也相同的四個骰子排成一列。

二、顏色全相同、且數字全相異的四個骰子排成一列。

三、顏色全相異、且數字全相同的四個骰子排成一列。

四、顏色全相異、且數字全相異的四個骰子排成一列。

我們計算的基準將排除擺放位置，單純以出現機率來計算。

一、顏色全相同、且數字也相同。

$$\frac{C_1^{16}\cdot 1\cdot 1\cdot 1}{C_1^{16}\cdot C_1^{16}\cdot C_1^{16}\cdot C_1^{16}}=\frac{1}{4096}$$

二、顏色全相同、且數字全相異。

三、顏色全相異、且數字全相同。

$$\frac{C_1^{16}\cdot 3\cdot 2\cdot 1}{C_1^{16}\cdot C_1^{16}\cdot C_1^{16}\cdot C_1^{16}}=\frac{3}{2048}$$

四、顏色全相異、且數字全相異。

$$\frac{C_1^{16}\cdot C_1^{9}\cdot C_1^{4}\cdot 1}{C_1^{16}\cdot C_1^{16}\cdot C_1^{16}\cdot C_1^{16}}=\frac{9}{1024}$$

分數期望值的計算，在遊戲中我們給狀況 1 給 5000 分，狀況 2 、 3 給1000分，狀況 4 給200分，因此期望值等於 5.9085 ，遠高於 0 ，就是希望玩家能從遊戲中得到成就感，最後為了簡化分數的數值於是所有數值除 50 ，也就是分數變成 100、20、4。

遊戲機率考察：

不論排列方式、先後，這邊考慮的是一次出4、8、12、16、20顆骰出現可以消去的機率。也就是已先知法去推測機率

骰子產生法：全隨機

分母皆為4096，格內數字是分子

	同色同字	同色不同字	不同色同字	不同色不同字	總合
4顆骰	1	6	6	36	49
8顆骰	2^3	6*2^3	6*2^3	36*2^3	49*2^3
12顆骰	3^3	6*3^3	6*3^3	36*3^3	49*3^3
16顆骰	4^3	6*4^3	6*4^3	36*4^3	49*4^3
20顆骰	5^3	6*5^3	6*5^3	36*5^3	49*5^3

表一：X 軸為算分規則、Y 軸為棋盤上的骰子數目

4.7、使用數學將程式簡化、提高效能

遊戲中較為麻煩的邏輯判斷地方骰子分數計算的判斷，每次計算都要以能夠拿最高分為準來計算，加上有特殊骰子（可以代表任何數字、任何顏色的骰子）的存在，所以要判斷的地方就更多且複雜，在經過幾次思考以後，把這部份特別拿出來轉化成代數表達式，並且簡化判斷的程序。

其中讓我減少最多行程式碼、以及減少最多判斷次數的就是：骰子數字全相異、數字也全相異的註狀況，我將這個情況轉化成代數式：

X + Y + Z + W = 34（我的骰子是使用 1~16 來代表，每個常數都代表一種骰子），其中 X 、 Y 、 Z 、 W 都代表一顆骰子，為何四顆骰子顏色相異、數字也相異會有這種特質呢？原因如果畫圖就很清楚了！如下表：

	數字 1	數字 2	數字 3	數字 4
黃	13	14	15	16
紅	9	10	11	12
綠	5	6	7	8
藍	1	2	3	4

表二、使用固定常數代表骰子

橫軸：骰子數字

縱軸：骰子顏色

假設今天我的 X 是藍色的數字 1 ，所以下一個就不能是藍色、以及其他只要數字 1 都不行，所以下一個我拿到的假設是綠色 2 ，所以這個 Y = 6 = 4 + 2，所以這時候數字 2 跟綠色也不能拿到了，假設下一個拿到紅色 3 ，所以我可以寫成 Z = 11 = 8 + 3 ，這時候最後一棵骰子一定要黃色 4 ，所以 Z 的代數表達是就是 Z = 16 = 12 + 4

這時候我確定如果顏色都不同、數字都不同的情況一定是：

（ 1 + 2 + 3 + 4 ）+（ 0 + 4 + 8 + 12 ）= X + Y + Z + W

因此用 X + Y + Z + W = 32 就可以過濾掉大部分不合格的判斷，再來就只剩下簡單的排除法了，那麼我們要排除什麼狀況呢？其實看上面的括號就可以知道是哪兩個判斷式了：

一、（1 + 2 + 3 + 4）代表：每個數字都必須不一樣，所以兩兩相減是不能成為 4 的倍數。

二、（0 + 4 + 8 + 12）代表：每個數字都必須在不同區間的顏色，所以必須判斷 1 <= X <= 4 且 5 <= Y <= 8 且 9<= Z <= 12 且 13<= W <= 16 ，分別代表 X 屬於藍色、Y 屬於綠色 、Z 屬於紅色 、W 屬於黃色。

4.8、遊戲計分動畫使用 Hash Table 資料結構

由於遊戲的邏輯性是存入一個 4 x 4 的陣列中，但是每次要去一一檢查「是否包含四顆骰子」，「如果已經達到四顆骰子，就要做一段計算分數的動畫，讓玩家知道遊戲正在算分」，而且還要去判斷「是否符合規則可以消除」；整個掃描表格的動作，其實是非常沒有效率的，因此我使用了 index 來存「目前已經有四科骰子」的是哪一個位置，那麼當我要計分的時候，只要利用 index 的索引值，去找到 Hash Table 在哪個位置，有哪些骰子要做動畫效果、要計分，於是就可以省略掃描整個 4 x 4 陣列（整個掃描總共要 19 次）的麻煩了。

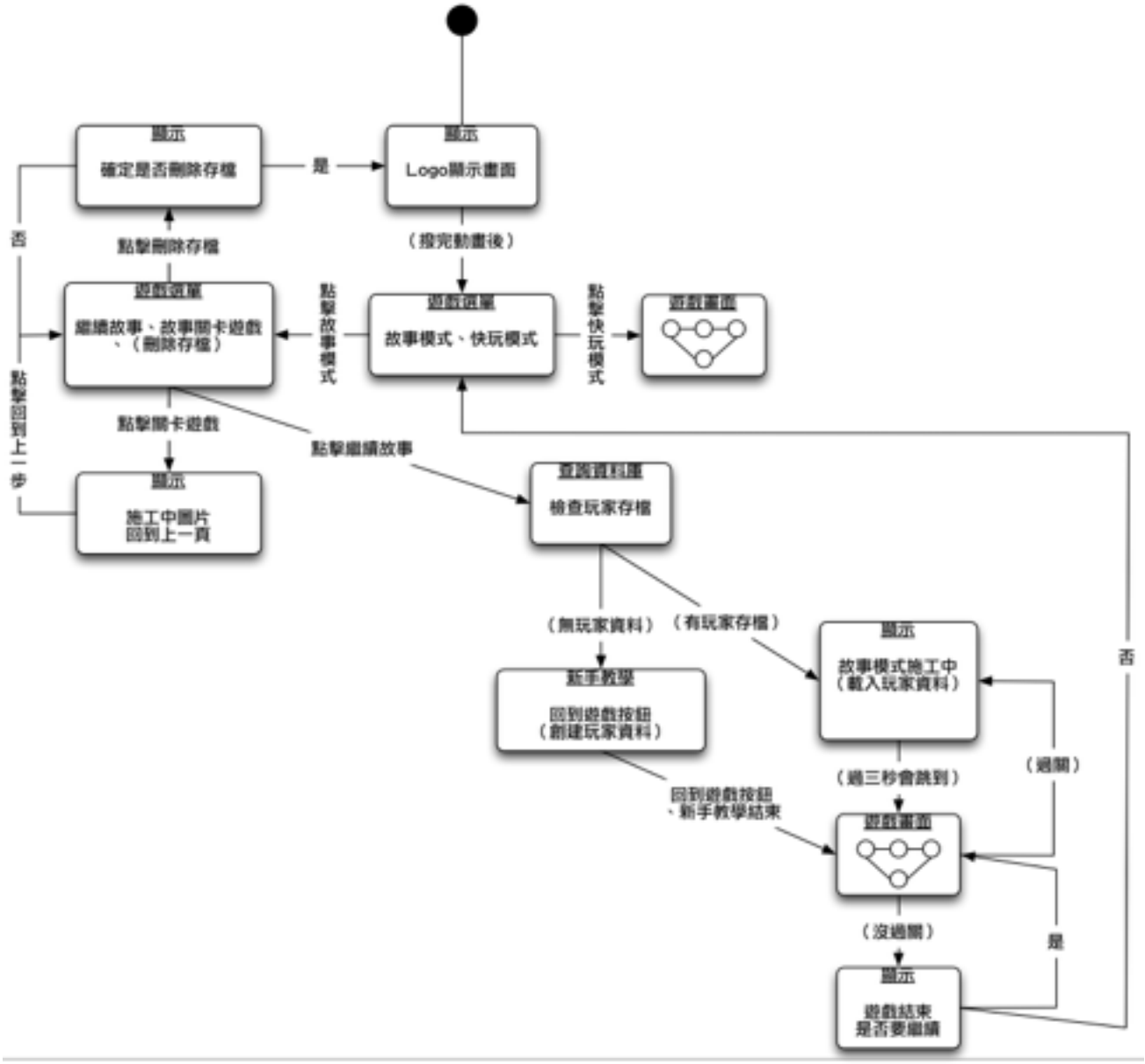
```
int havingFourDiceHashNumber[19];
```

用來存放達到 4 顆骰子的 index 索引值。

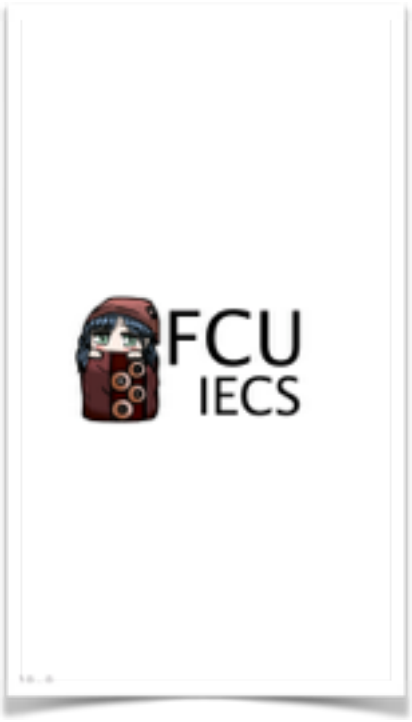
```
int hashGameBoard[19][6];
```

當我存放索引值的地方有數值了以後，我就會利用這個數值去hashGameBoard[索引值][6]; 找是哪些骰子要做動畫效果，並且邏輯判斷是否這些骰子符合遊戲規則可以消除，由於遊戲本身棋盤式不會在擴充的，所以我不用擔心 Hash Table 會有碰撞的問題。

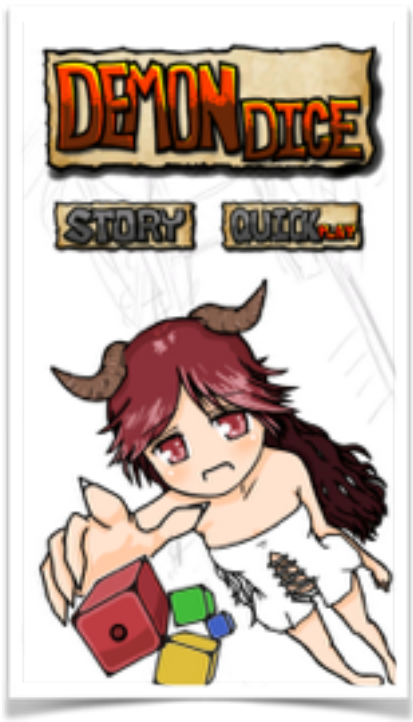
五、STD圖、遊戲畫面流程



圖二十五、遊戲畫面的STD圖

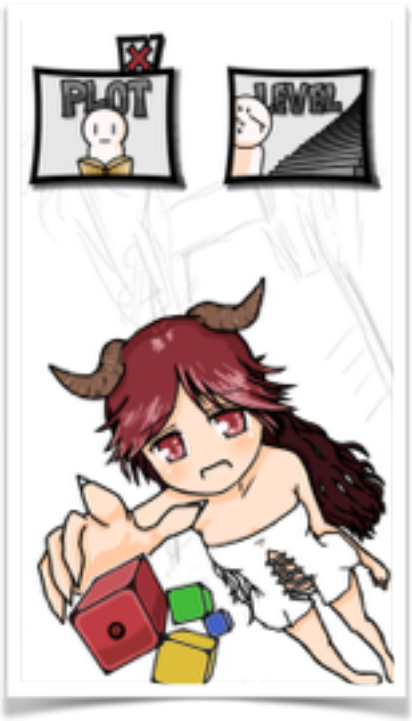


圖二十六、Logo畫面



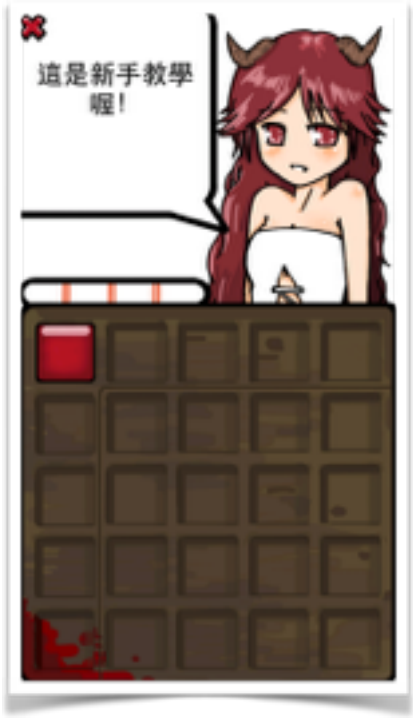
圖二十七、主選單畫面

故事模式、快玩模式

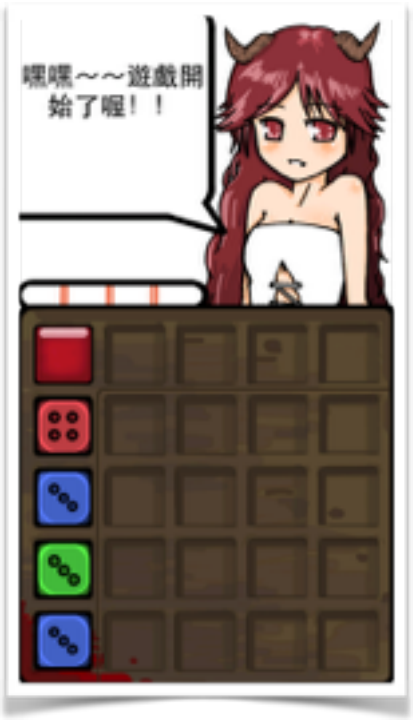


圖二十八、故事模式選單畫面

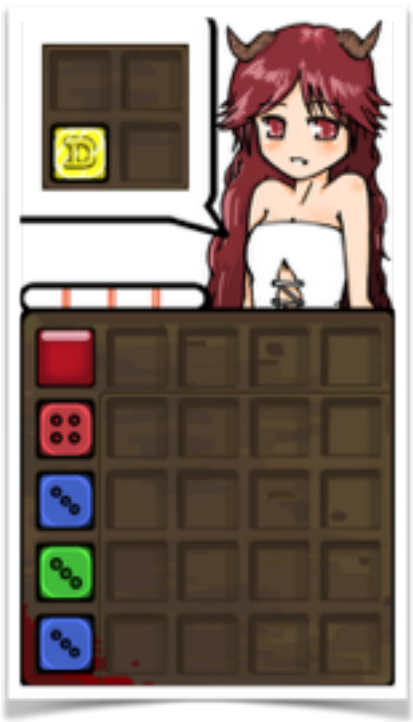
繼續故事、關卡模式



圖二十九、新手教學畫面



圖三十、主要遊戲畫面



圖三十一、魔王超級骰子

可以代表任何骰子的超級骰子

圖三十二、設定畫面

相本模式、技能模式

新手教學、遊戲設定

圖三十三、技能畫面

圖三十四、未完成關卡模式的施工中畫面

六、使用性測試

為了讓遊戲軟體能夠經得起市場的考驗，於是我們著手進行使用者測試，讓玩家能真實反應出他們玩魔王骰子遊戲時，所遇到的問題，並且將玩家遇到的問題、以及他的感受進行記錄、分析，然後再來修改遊戲的設計以及流程。

我們將附錄中的表（附錄一），給逢甲資訊工程系的六位學生測，年齡介於 18 ~ 23 歲之間，底下是我們統計後的結果：

問卷結果統計

統計對象 : 資訊系學生 * 6

背景調查						
有使用iPhone的經驗						3
有玩過很多種類的手機遊戲的經驗						3
	非常好 = 5	還不錯 = 4	普通 = 3	略差 = 2	很糟 = 1	平均分數
1.	1	2	3			3.67
2.	1	4		1		3.83
3.	3	3				4.5
4.	1	3	2			3.83
5.	1	2	2	1		3.5
6.	1	2	2	1		3.5
7.	2	4				4.3
8.		3	3			3.5
9.	3	3				4.5
10.	1	1	4			3.5
11.	1	2	2	1		3.5
12.	1	2	2	1		3.5
13.	1	2	2	1		3.5
14.	1	3	2			3.83
15.	3	0	3			4
16.	2	1	3			3.83
17.	3	3				4.5
18.	2	2	2			4
19.	3	1	2			4.17
20.	1	4	1			4
21.	0	2	0	4		2.67

BUG與意見回報彙整(條列式)

七、心得

2011年昇大二的暑假，接觸了一款結合傳統 RPG （打怪升級）以及 FPS （第一人稱射擊）的創新遊戲，讓我費寢忘時的沉迷好幾天，於是就萌生了想自己寫一款結合卡片遊戲以及其他類型遊戲元素的創新遊戲，但是當時的我能力不足，也不懂系統分析與設計、人機介面，甚至連技術方面也只懂 C 語言以及簡單的物件導向，更不用說物件導向更深入的設計模式了，雖然有很好的構想、並且野心勃勃的去跟很多人分享我的想法，卻被指出這個遊戲的專案能成功實現的機率很低，原因在於：專案太過龐大、很多細節沒有詳細的內容、實作上要用什麼樣的工具還是個問號，另外專案管理的經驗不足，所以後來我將目標轉小，轉為實作一個規模較小的觸碰遊戲，就是現在的專題「魔王骰子益智遊戲」，而遊戲內很多目標是為了實現更龐大專案而先嘗試、實作的，比方說：觸碰物件讓物件移動（骰子的移動）、或是遊戲內角色的對話設計成容易擴充、且能夠經由判斷來改變其對話的方式來設計的（使用了SQLite技術），2011年開始，我修了系統分析與設計課程，了解到軟體專案管理跟一般的管理的差異，比方說軟體是無形的，不像一般工程是可以觀察到實體而知道他的進度；另一方面也學到了敏捷開發法，讓我更了解使用者與設計者之間的溝通的重要性；另一方面我也閱讀了 iOS 的原生語言 Objective - C 的書籍，並且動手嘗試實作一些簡單的小遊戲；當原生語言 Objective - C 越來越熟悉的時候，我開始閱讀更深入的書籍，比方說 Open GL ES 於 iOS 上開發遊戲的書籍，還有 cocos2D for iPhone 的相關書籍，這時候我一邊閱讀書籍一邊實作，讓能力不斷進步， 以前一些看不大懂得書籍也因為修了「物件導向軟體工程」、以及「資料庫」後能看的懂其內容，加上團隊中增加了一個熟習資料庫的同學，整個遊戲對話系統就使用資料庫的方式實作了起來，並且更進一步的將對話以事件的方式包裝起來，未來可以輕易的加入新的事件以及加入新的 NPC ，讓遊戲模組化以及可攜性提高。

製作遊戲牽涉到的層面很廣，包含專案的管理、美術設計、遊戲的風格以及都遊戲的背景，甚至軟體所使用的技術、或是資料結構，我們從中學到了許多、也解決了很多難題，遊戲專題雖然是條不好走的路，但是帶給我們的成就感是無可取代的！

八、參考文獻

- 精通 Objective-C 程式設計, 5/e (Programming in Objective-C, 5/e) Stephen G. Kochan 著、蔡明志 譯，碁峰
- Learn cocos2d Game Development with iOS 5 , Steffen ltterheim 、 Andreas Low 著，Apress
- Cocos2D 遊戲程式開發供略 - 動手撰寫你的第一支 iOS 遊戲 , Rod Strougo , Ray Wenderlich 著、蔡明志 譯，碁峰
- 人月神話 - 軟體專案管理之道 , Frederick P.Brooks, Jr. 著 , 錢一一 譯，經濟新潮社
- 系統分析與設計 - 理論與實務應用 , 吳仁和、林信惠 著，智勝
- INTERACTIVE DESIGN 人本互動設計 , ANDY PRATT & JASON NUNES 著、吳國慶 譯，果禾文化
- iPhone 遊戲開發之練 , Peter Bakhirev | PJ Cabrera | Ian Marsh | Scott Penberthy | Ben Britten Smith | Eric Wing 著、羅友志 譯
- CocoaChina 蘋果開發中文站 （http://www.cocoachina.com ）
- Stack OverFlow（http://stackoverflow.com）

九、使用性測試報告

使用者滿意度暨BUG回報

採用隨機測試(HALLWAT TEST)

評判標準根據Nielsen's Usability Heuristics 的十個建議

我是_____　我有使用iPhone的經驗 口　　我有玩過很多種類的手機遊戲 口

5=非常好 4=還不錯 3=普通 2=有點糟　1=很糟

系統狀態可見性

透過在合理時間內的合適回饋，系統應該讓用戶瞭解目前的狀態。

內容	5	4	3	2	1
1.我隨時都知道目前的系統狀態是忙碌、閒置或可操作狀態					
2.我覺得系統的反應時間恰當，沒有疑惑的狀況產生					

系統與真實世界的關聯性

該系統應該以使用者熟悉的語言、文字、詞彙與概念來呈現，而不是使用系統導向。

內容	5	4	3	2	1
3.我能夠了解遊戲內的出現的文字、圖案、按鈕與劇情內容					
4.我可以很快熟悉遊戲的操作方式和按鍵配置					

使用者的控制度與自由度

使用者時常以嘗試的心態來操作系統功能，他們需要一個明顯的「離開系統」來離開使用者不需要的狀態。應該支援復原步驟與重複步驟。

內容	5	4	3	2	1
5.我可以隨時中斷遊戲、離開遊戲或是進行其他選項					
6.我能夠很快理解遊戲模式和特殊選項的操作					

一致性和標準

使用者不應該猜測同一種動作是否使用不同的字彙、狀態或動作

內容	5	4	3	2	1
7.我認為遊戲中的操作具有一致性和唯一性					

8.我覺得遊戲的玩法、規則和目標很明確					

預防錯誤

這是比錯誤訊息還要親切的設計，『預防』是發生問題最先要考慮的事情。不管是移除容易出錯的的條件，或是讓使用者確認他們接下來要做的行動皆是。

內容	5	4	3	2	1
9.我可以透過新手關卡來快速學習遊戲					
10.我在正常操作下覺得很順利，沒有遇到錯誤					

讓使用者去認識系統，而非去記憶

儘量減少使用者需要記憶的事情、行動以及可見的選項。使用者不應該記憶太多步驟。系統使用說明應該在適合的地方表現的顯眼且可輕易使用。

內容	5	4	3	2	1
11.當我需要幫助時，我可以快速找到					
12.我不用記憶太多設定和規則也能進行遊戲					

靈活性與使用效率

應該有彈性讓初用者和慣用者都能方便使用。例如慣用者可以使用快捷鍵來提昇他們的使用速度; 允許使用者設定常做的動作; 畫面可以延伸等。

內容	5	4	3	2	1
13.我認為操作設計上，可以幫助我提升效率					
14.我認為內容設計上，可以幫助我提升效率					
15.我認為UI設計上，可以幫助我提升效率					

美術與簡化設計

對話框不應該包含無關緊要或很少用到的訊息。對話框的每一個額外的部份都會相對地降低主要資訊的顯眼程度。

內容	5	4	3	2	1
16.我覺得UI設計具有美感					
17.我覺得UI設計方便簡潔					
18.我覺得遊戲美工、音效、對話系統，能夠提升我對遊戲的興趣					

幫助與說明文件

即使是最好的系統也不能沒有說明文件，系統也需要提供幫助與說明文件。說明的資訊應該很容易被找到。

內容	5	4	3	2	1
19.我能夠很簡單的找到遊戲說明					
20.我覺得遊戲內說明，能夠解決我的困惑					

幫助用戶認識、偵錯並從錯誤中恢復

錯誤訊息應該以敘述文字呈現，而不是錯誤代碼，並且精確地指出問題以及提出建設性的解決方案。

內容	5	4	3	2	1
21.當遊戲出錯時，我知道該如何處理					

感謝你協助這次的測試，在這次的測試中如果你發現了任何BUG或是有任何改進的建議請寫在空白處或告知我