東 南 科 技 大 學

數位遊戲設計系

專 題 製 作

城堡

CASTLE

學生　：溫浩鈞　40625005

李勁霆　40625014

王韋凱　40625017

陳治豪　40625022

郭哲瑜　40625059

指導老師　：謝瑞宏　老師

中 華 民 國　109年　12月

目錄

第一章 企劃目標

第一節 研究目的與動機

目的

想做出一款簡單並以 FPS 為主要核心玩法的 Roguelite 遊戲

動機

市面上有許多 Roguelite 類型遊戲，但很少以第一人稱呈現
本組成員皆喜歡 Roguelite 類型
本組成員對 FPS 類遊戲製作有一定經驗

第二節 遊戲基本概述

遊戲前導

啊！你好啊。
你是新來的主管吧，這裡是討伐與解析異常的單位
我們是與異常接觸的第一線，前任主管達成了工作項目被調去了別的單位
接下來我會告訴你要做什麼，以及要怎麼做
讓我們一起漂亮的完成工作吧。

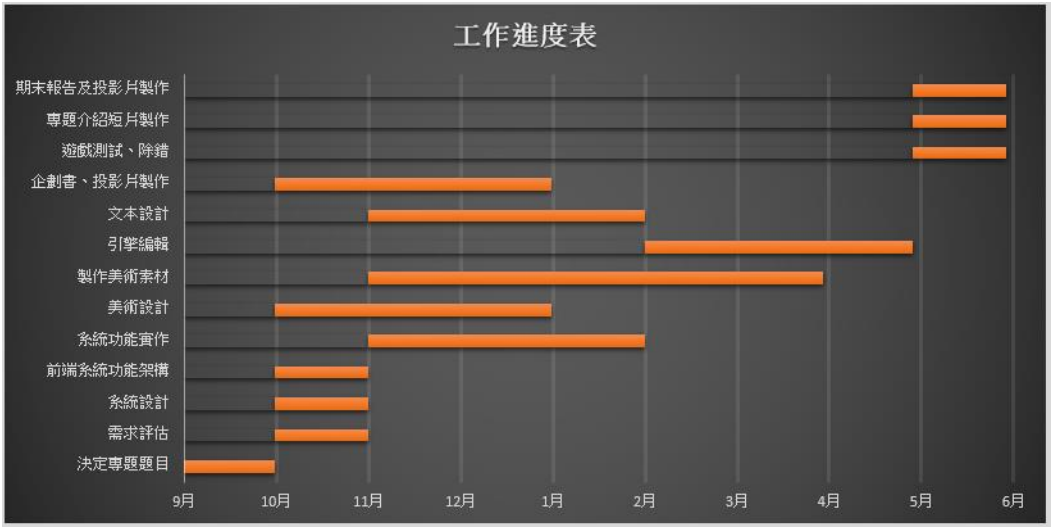
遊戲大綱

這是一款融合了第一人稱射擊、Roguelite 元素和 RPG 策略選擇的冒險遊戲，特色為可以隨意搭配多種裝備創造玩法，獲得強大力量的遺物在隨機性關卡中進行冒險挑戰。

在遊戲中，玩家每次的關卡體驗都是隨機的。每一次重新開始遊戲都是新的體驗。你可以在不同的關卡體驗中遇見不同的裝備、遺物、不同的關卡路線和戰鬥節奏。

第三節 專題實施進度表

決定專題題目	109年9月	109年10月	30
需求評估	109年10月	109年11月	31
系統設計	109年10月	109年11月	31
前端系統功能架構	109年10月	109年11月	31
系統功能實作	109年11月	110年2月	92
美術設計	109年10月	110年1月	92
製作美術素材	109年11月	110年4月	151
引擎編輯	110年2月	110年5月	89
文本設計	109年11月	110年2月	92
企劃書、投影片製作	109年10月	110年1月	92
遊戲測試、除錯	110年5月	110年6月	31
專題介紹短片製作	110年5月	110年6月	31
期末報告及投影片製作	110年5月	110年6月	31



第四節 工作分配

人員	工作項目
溫浩鈞	企劃書撰寫、資料查找、遊戲測試，模型製作、人物設計、遊戲測試、
李勁霆	程式撰寫、資料查找、遊戲測試、專案統整、UI設計
王韋凱	模型製作、渲染特效、音效設定、動畫製作、遊戲測試
陳志豪	模型製作、遊戲測試、資料查詢、會議記錄
郭哲瑜	企劃書撰寫、資料查找、遊戲測試、模型製作、對話設計、數值設定、影片製作

第二章 參考文獻與分析

第一節 目標市場分析

目標平台:STEAM：

STEAM 是目前最多用戶的遊戲平台，上面也有許多獨立製作遊戲，十分適合讓我們的作品上架，雖然有其他專門給獨立遊戲，或是抽成更少的平台，但 STEAM 龐大的用戶群以及穩定的經營依舊是業界上架首選。

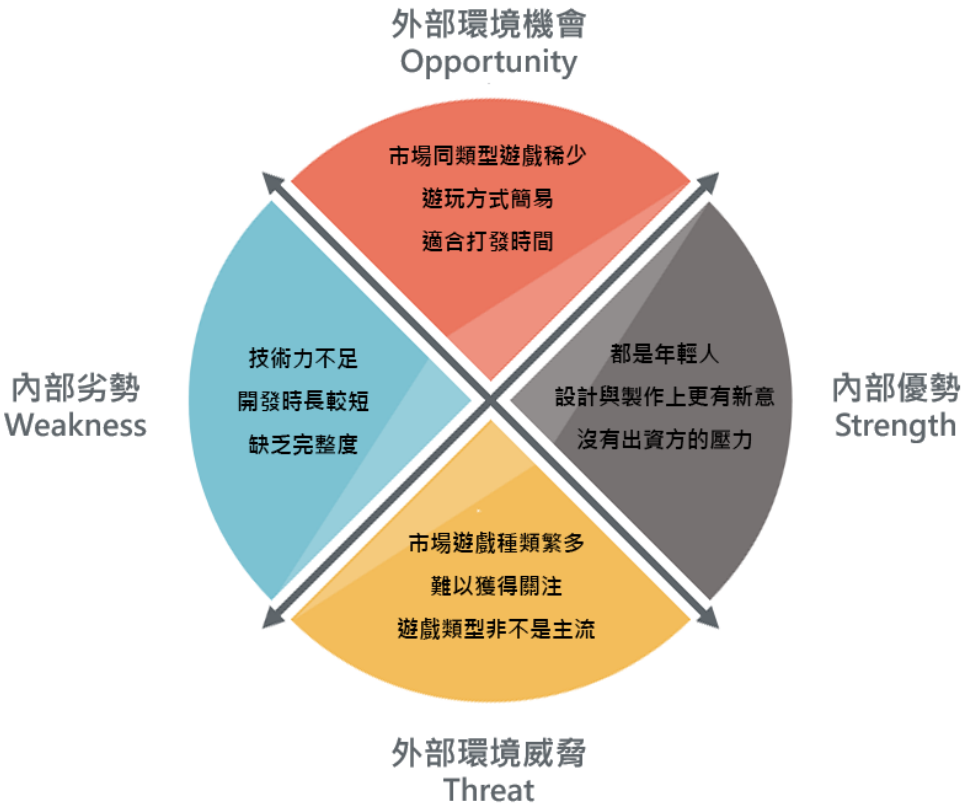
目標玩家:中度 PC 玩家：

中度 PC 玩家大多願意嘗試各式遊戲，不向重度會花大量時間於1~2款喜愛的遊戲，這樣的客群來遊玩我們的遊戲相對較高，而我們的遊戲簡單上手，但又有足夠的遊戲內容，能夠滿足中度玩家所需。

目標類型:Roguelite，單機 FPS：

只有一個問題，市面上這兩類合併遊戲種類極少，沒有太多遊戲可參考但相對代表著市場並沒有同類型競爭者，十分有利。

第二節 SWOT 分析



第三節 開發軟體

設計草圖與2D 美術圖皆使用 Photoshop
3D 模型建模為 Maya 和 Blender
貼圖繪製 Substance Painter

美術

設計草圖與2D 美術圖皆使用 Photoshop
3D 模型建模為 Maya 和 Blender
貼圖繪製 Substance Painter

程式

遊戲程式 Visual Studio C#
遊戲引擎: Unity 2020.1.0f1 (64-bit)

文件

Word、Excel、PowerPoint

第三章 遊戲概述

第一節　遊戲世界觀與故事

本遊戲劇情在玩家視角是碎片式的架構，以下為原設定，玩家能了解一部份但不完全。

遊戲故事綱要

玩家是一名實驗室上級主管，因為[redacted]收容突破，而被派遣到此與[白鼠]部隊共事，派遣有如工具般的工作人員，以性命為代價進行調查，獲得升級實驗室的資源，及更有效的使用異常方法。

在遊玩的過程玩家會有機會獲得各式故事線索，像是 NPC 們過去或現在的故事，怪物們存在的原因，裝備的由來及設定等等。玩家要自己透過自己所獲得的線索，去拼湊遊戲的世界觀及故事。

怪物故事綱要

所有異常都是世間的傳說與謠言而聚成的，就像是日本"萬物皆神靈"的概念一樣。

而也正因為這樣諸多不同形式不同概念的異常就這樣出現了。像是，Slender Man，他是一個身材瘦長的男人，常穿著襯衫，打著領帶，衣著整潔，但令人毛骨悚然的是他沒有臉。會使受害者陷入催眠狀態，然後神不知鬼不覺地將受害者帶走。通常在寂靜無人的荒郊野外或是深林深處出沒，將落單的人們殺掉。

起初這只是來嚇唬那些不聽話的小孩子。而沒人知道這項傳說因為他們的謠言而成了真實。

遊戲內會有各式這種故事及設定，給玩家去探索。

組織故事綱要

起初只是一些喜歡超自然現象的瘋子的集會，但這世上就是有常理無法解釋的東西，而正好就被他們找到了，他們如發狂般做著實驗與研究，用得到了理論或是產物去挖掘發現更多異常，隨著時間的推進從原先小小同好會發展成握有顛覆一切力量的神秘組織。

組織有數個部門與單位，而玩家遊戲中所在的是[白鼠]，負責處理建築異常和封閉空間異常的部隊，職責是調查、探索、收容地下或封閉區域內出現的異常現象，基本上是全組織陣亡率最高的單位，也因為這樣他們的調查方式與其他部隊不一樣，是利用各種方式招來的人力搭載紀錄、分析裝置，並以命為代價去調查(即便存活身上裝置也會自爆)，原先被此提案被駁回，但以1.這些調查人員遲早會死 2.這些外部來的人沒有機會將資料帶出 這兩項理由而被通過。

事實上這次的收容突破是人為的，但組織所有人都以為只是收容出錯，而隨著[白鼠]的調查，會發現一些被組織隱藏起來黑紀錄，也會發現這起事件的幕後黑手。

白鼠

第二節 遊戲對話

在遊戲中可以與主畫面的 NPC 對話，瞭解組織部分秘密，或是更加了解該 NPC

第三節 遊戲特點

遊戲上是 Roguelite 的特色，永久死亡與大量隨機要素，但又有一定程度的成長，避免永久死亡過大的挫折。

設定上會採用現實中熱門的都市傳說作為題材來設計怪物，玩家一邊探索不可預期實驗室，一邊揭露組織與怪物的秘密。

第四節 系統規格

最低配備:

需要 64：位元的處理器及作業系統
作業系統：Windows 7 or newer, 64-bit
處理器：Intel Core i3-6100 / AMD FX-8350
記憶體:：4 GB 記憶體
顯示卡：GTX 580 / AMD HD 7870
DirectX： 版本：11
網路： 寬頻網際網路連線
儲存空間： 4 GB 可用空間

建議配備:

需要 64：位元的處理器及作業系統
作業系統: ：Windows 7 or newer, 64-bit
處理器： Intel Core i5-4670K / AMD Ryzen 5 1500X
記憶體： 4 GB 記憶體
顯示卡： GTX 680 / AMD HD 7970
DirectX： 版本：11
網路： 寬頻網際網路連線
儲存空間： 4 GB 可用空間

第四章 遊戲機制

第一節 遊戲類型

Roguelite 類第一人稱射擊；俗稱爬塔遊戲或農 GAME。

第二節　遊戲操作

基本操作:

甲、移動:

i.　前、後、左、右，對應 W、A、S、D 鍵。

ii.　跳躍，對應 SPACE 鍵。

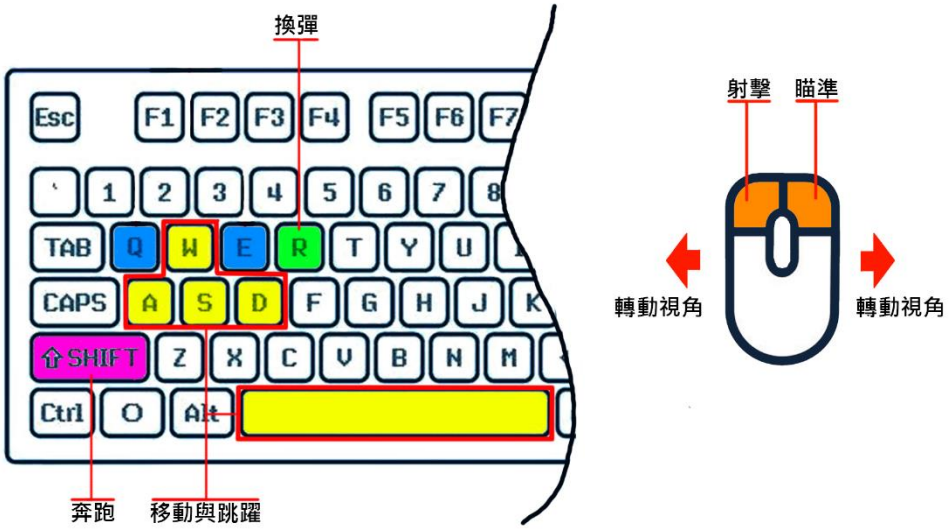
乙、攻擊:

i.　開槍，對應滑鼠左鍵。

ii.　瞄準，對應滑鼠右鍵。

iii.　換彈，對應 R 鍵。

iv.　奔跑，對應 SHIFT。

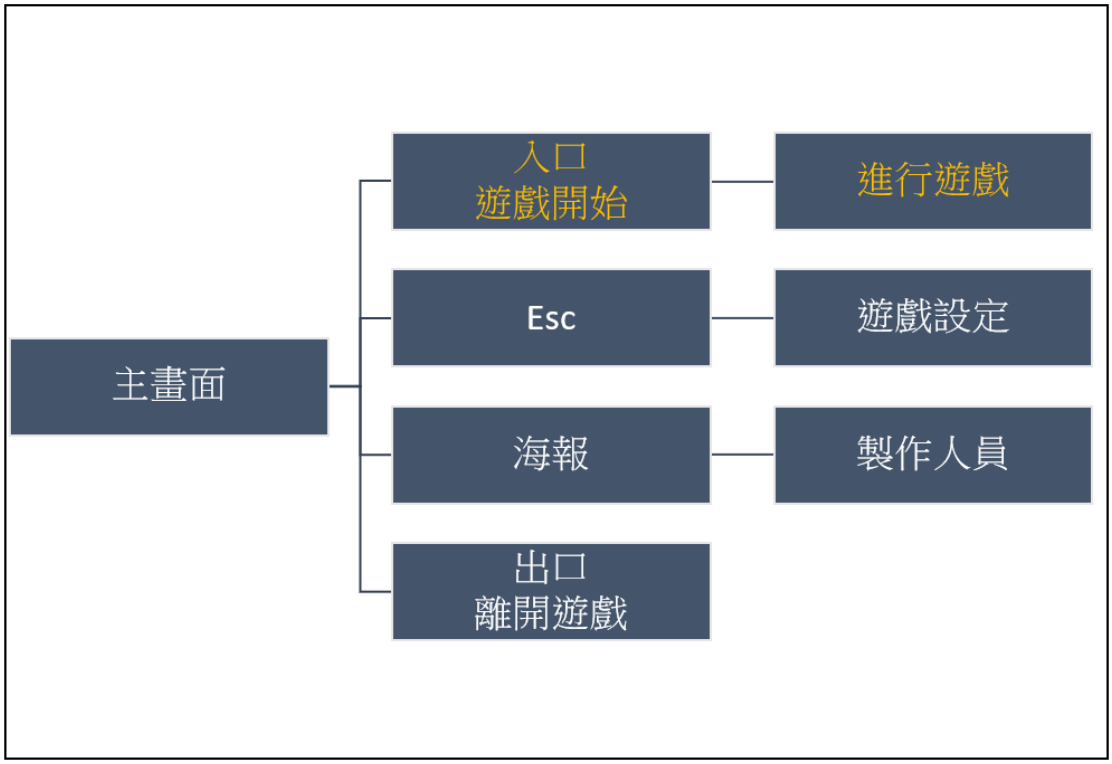


第三節　遊戲方式

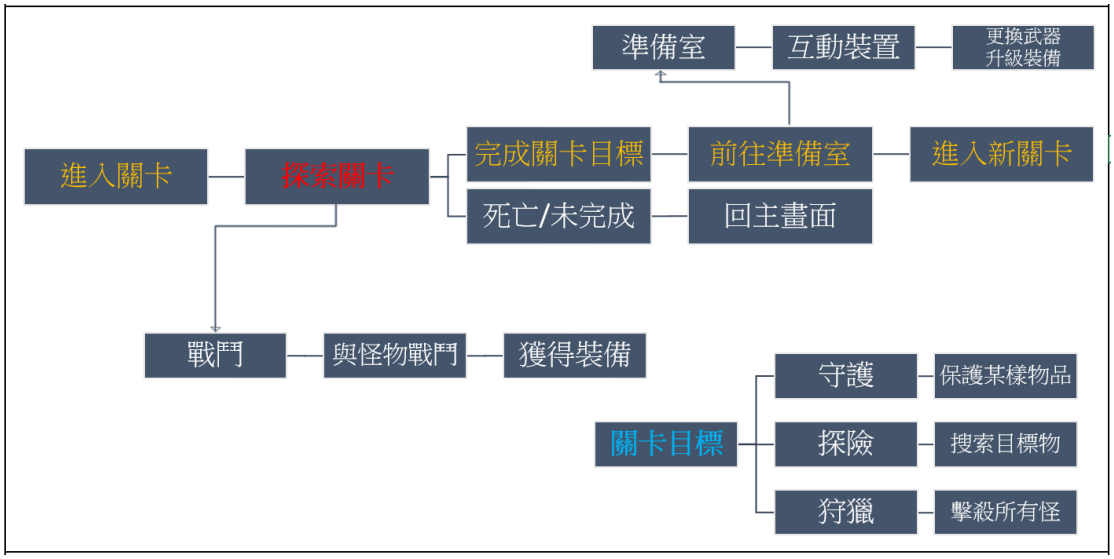
在遊戲中反覆執行，進入關卡→完成任務與擊殺怪物→進入新關卡

第四節　遊戲流程圖

主畫面遊戲流程



遊戲流程圖



第五節　關卡設定

甲、關卡數量為無限關，只要玩家不死達成關卡任務就能前往下一關。

乙、關卡

1. 怪物會在玩家進入關卡後不斷生成，即便任務完成。
2. 入口等於出口，出入口在一開始進來後便會上鎖
3. 門開啟後地圖上所有裝備會自動轉為金錢給予玩家
4. 解鎖方式為三種
   i. 守護，保護地圖上的某個物件一定時間
   ii. 探險，在地圖尋找特定物件
   iii. 狩獵，殺光地圖所有怪物

丙、遊戲難度會隨著遊玩時間增強。

1. 增強為給所有怪物更多血量與傷害

丁、準備室內會有可互動裝置

1. 武器交換機
   i. 交換武器
2. 升級裝置
   i. 把戰鬥中獲得的裝備升級

第六節 數值設定

1. 所有單位共用數值
   甲、血量
   乙、護甲
   丙、護甲恢復速度：護甲每秒恢復數量
   丁、護甲延遲時間：護甲未受到攻擊後特定秒數才開始恢復
   戊、移動速度：分為一般走路與奔跑
   己、傷害：根據攻擊方式不同會有不同計算方式
   庚、攻擊速度
2. 玩家特有數值
   甲、經驗值：玩家升級判斷依據，獲得方式為造成傷害
3. 怪物特有數值
   甲、掉落裝備類
   乙、掉落裝備率
   丙、強度等級:隨著遊玩時間以及關卡數增加該數值
4. 道具數值
   以下數值為道具可以修改的數值種類
   甲、血量
   乙、護甲
   丙、移動速度
   丁、奔跑速度
   戊、換彈速度

5. 武器數值
   甲、傷害
   乙、射速
   丙、彈夾
   丁、瞄準速度
   戊、換彈速度
   己、後座力
   庚、貫穿力

第七節 聲音與音效

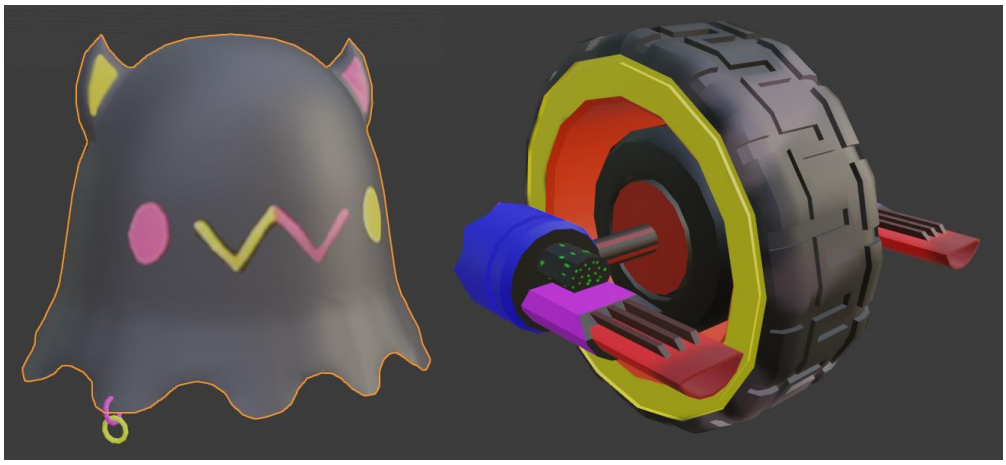
參考網址

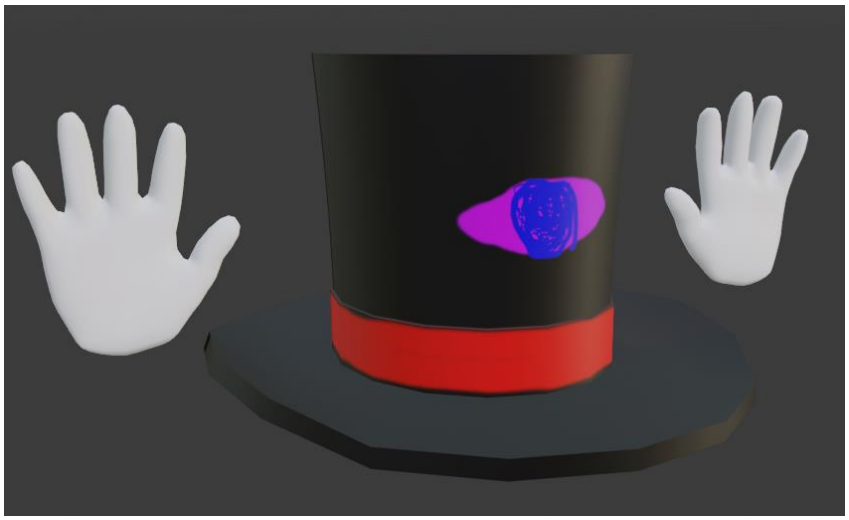
小森平的免費下載音效：
https://taira-komori.jpn.org/freesoundtw.html

freesound：
https://freesound.org/

第五章 遊戲美術

第一節 怪物





怪物名稱:闇鬼 / 火胎 / 魔術帽

背景故事:

原先都是生活中的小東西，像是玩偶.輪胎.帽子這樣，但因為持有者的執念發生了異變，最後被帶回收容於實驗室，被嘗試複製出更多樣本，一邊調查異變原因一邊調查能使用於何處。

- 所有的小怪都是平凡之物發生異變之後被收容

第二節 NPC

角色名稱: 角鯨

角色年齡: 不明 / 性別: 雙性 / 個性:熱情、不拘小節、神秘

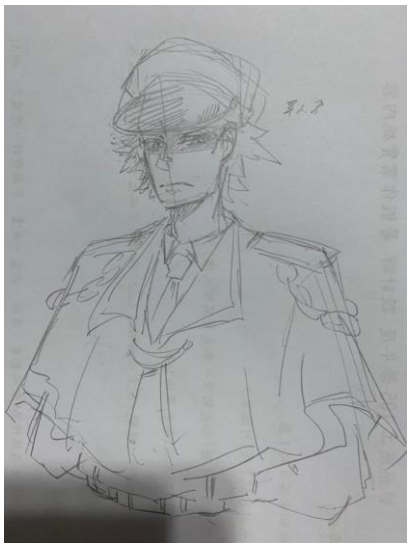
角色功能: 系統設定。

背景故事:

早期發現的異常之一，詭異強大且危險，是[redacted]博士於[redacted]峽谷探勘時偶然發現的，一個在巨大縫隙中並散發著淡淡藍光的水晶，裡面沉睡著最初的角鯨。

組織接連派出了軍隊前去收容，但都以失敗告終。而在最後的全軍要被消滅時，偶然看到了指揮所的電腦中的東西倍感興趣，而自願被收容與實驗，作為交易給予實驗室一級以下資料閱覽全與以不透漏組織機密為前提的自由，隨著組織的發展壯大，他也受到組織的認可與信任，而擔任資料處理與新兵訓練的職位，身上的機械零件，一部份是修復當初收容時的戰損，一部份是為了更有效的與組織系統連接管理。

- 這次會來到[白鼠]正是因為知道了某些事，而主動前來



角色名稱: 米勒[maillor]

角色年齡: 34 / 性別: 男性 / 個性:冷靜、理智、孤獨

背景故事:

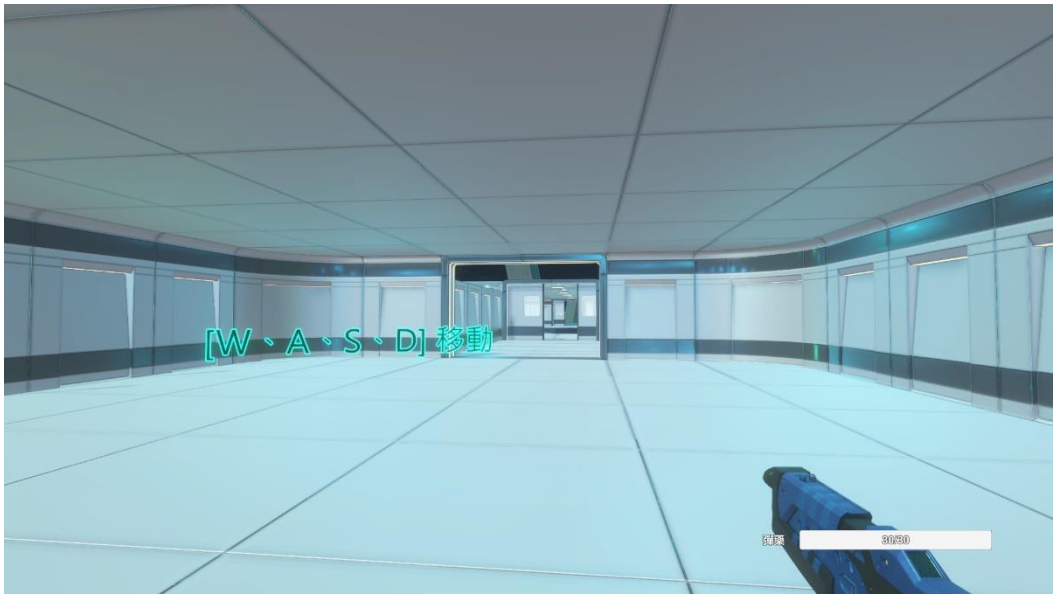
強大認真的隊官，多次參與過各式作戰，是新上任[白鼠]長官，雖然知道被調到這裡就是等死，但並不大算輕易死掉，他堅信這次收容意外是個機會，他清楚知道自己還有一些必須完成的事。

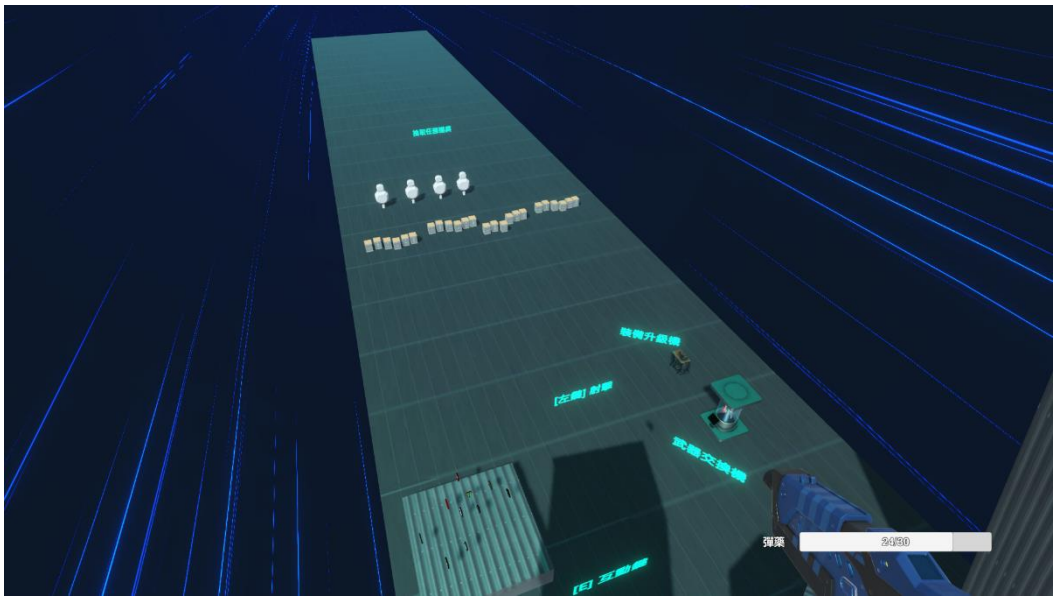
在學生時期與弟弟一同在海邊遊玩時意外看見收容現場，很幸運的他們並沒有被發現，從那刻起兄弟倆變對超自然現象有興趣，起初一起成為研究員，直到他們面對真正的異常，在那絕對的力量之前，米勒清楚知道自己要變強”保護弟弟”，也因為這件事他倆被組織看上並且加入了，弟弟繼續鑽研研究與解析努力，而米勒則加入正規部隊受訓。

- 弟弟最後所待的研究所就是這次事發地點

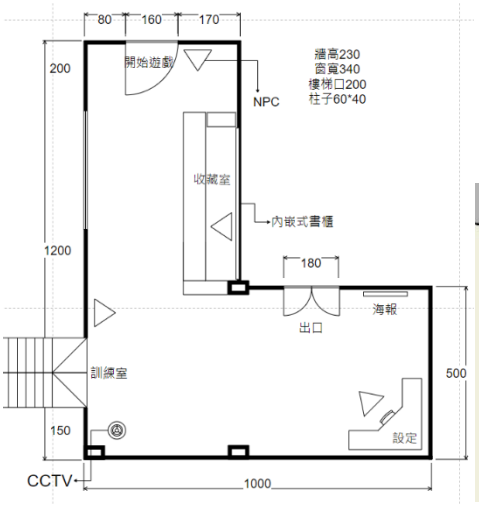
第五節　場景

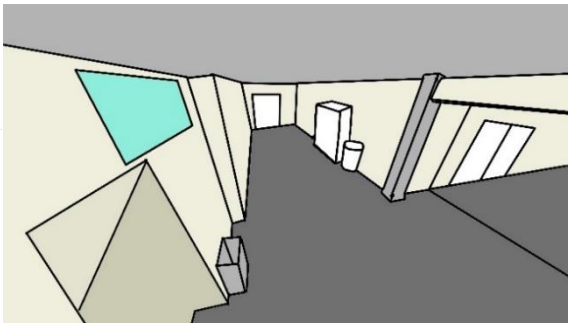
新手教學場景





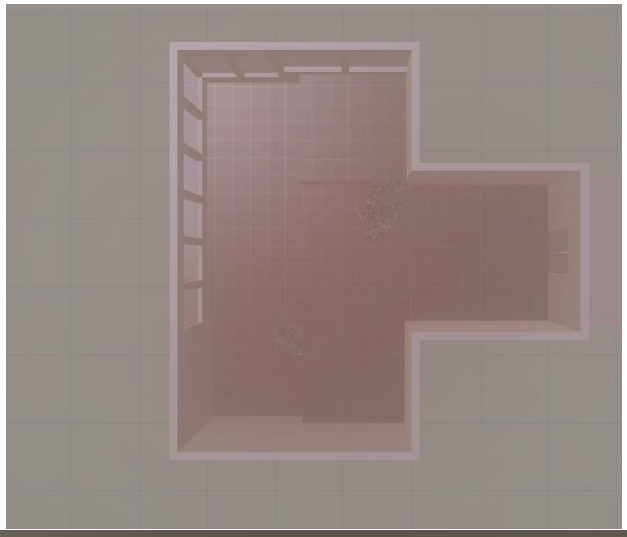
主場景

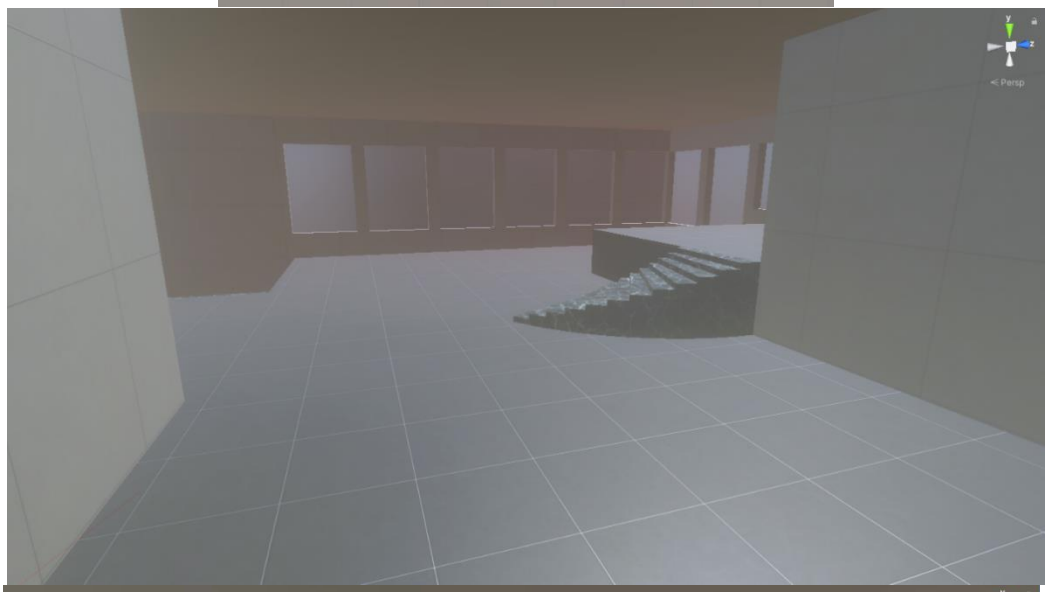

80
160
170
200
開始遊戲
NPC
牆高230
窗寬340
樓梯口200
柱子60*40
收藏室
內嵌式書櫃
1200
180
出口
海報
訓練室
500
150
設定
CCTV
1000

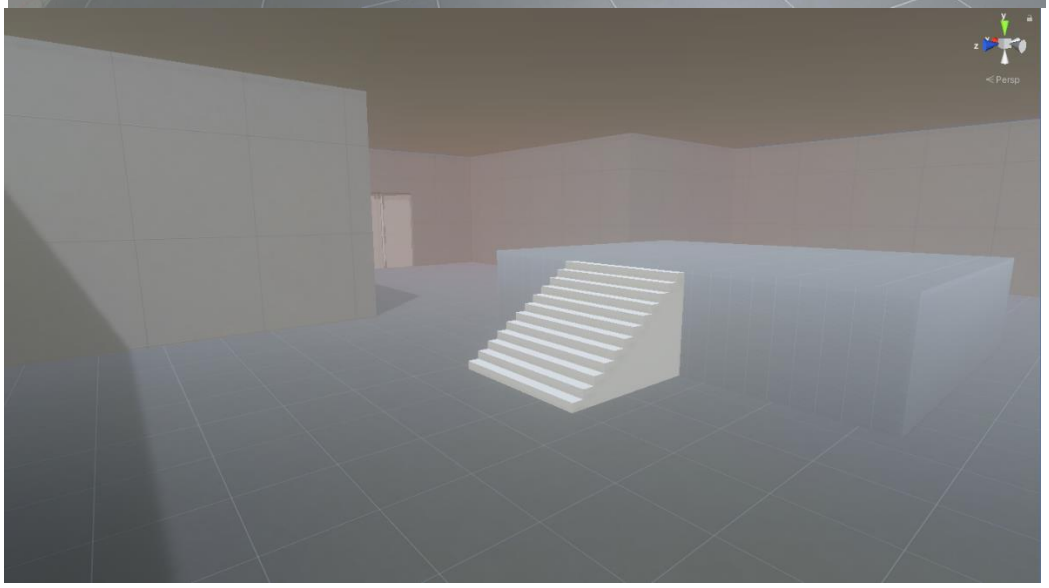



Miller
互動物件切換：D、A，互動案件為：E

戰鬥畫面場景







第六節 道具

在這邊會介紹各類道具的特性與能力。

武器

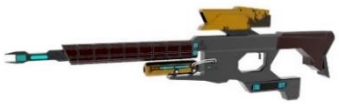
大多武器都有屬於自己的特色，能擁有特殊的攻擊方式或是機制
簡單的有普通步槍，高速連發的雙槍
或是射速會隨著命中敵人次數上升的機槍，不必瞄準會自己命中的手槍
武器還有而外效果，像是命中敵人對方無視護甲，開槍兩倍傷但無法開鏡
武器:

ID	名稱(中文	名稱(英文	傷害	射速	彈量	腰射擴張速	瞄準速度	填裝速度
1001	卡賓步槍	M4A1	75	7	30	1.5	4	2.5
1002	波波沙	PPSH41_G	25	15	51	5	6	5
1003	重砲	SniperRifle	200	2	10	10	3	5
1004	小衝鋒手槍	DualWieldi	50	10	30	10	6.5	1
1005	卡拉希尼槍	AK74M	75	7	30	1.5	4	2.5
1006	等離子衝鋒	Plasma	50	10	30	10	6.5	1
1007	科幻砲	SciFiRifle	100	2	15	10	1	5
1008	鐵手槍	IronPistol	150	15	15	10	10	2
1009	雙鼓式霰彈	DoubleDru	10	5	40	2	4.5	4
1010	三連衝鋒槍	Tec_9	50	5	25	2.5	6.5	1







裝備

裝備通常有明確強化方向，主要分為攻擊與防禦，但也有著兩者平衡或是給予奇怪能力的裝備

攻擊特化：增加大量傷害或射速，給予一些增傷或是擊中效果的能力

防禦特化：增加大量護盾或血量，給予一些減傷或是而外回血的能力

特殊：並不特別給予數值，獲得較為特殊的能力，多一條命、空中跳躍之類

ID	名稱(中文	名稱(英文	血量	護盾	移動速度	跑步速度
1001	Shot	Shot	100	0	0	-2
1002	可樂	nul	30	0	1	1
1003	平底鍋	nul	0	20	0	-1
1004	白羽扇	nul	0	20	1	2
1005	吊嘎	nul	-10	-10	1	2
1006	西瓜扇	nul	50	30	0	-1
1007	香爐	nul	100	50	0	0
1008	梳子	nul	30	30	0	0
1009	鈕扣	nul	10	10	1	1
1010	黃金鎧	nul	20	80	0	-1
1011	綠帽	nul	-10	80	0	1
1012	維京人頭盔	nul	160	20	0	-1
1013	斗篷	nul	20	20	1	0
1014	布甲	nul	30	50	5	0
1015	戒指	nul	20	60	0	0
1016	翅膀	nul	0	0	1	2
1017	彈夾	nul	0	10	0	0
1018	魔法帽	nul	90	-20	1	1
1019	惡魔盒	nul	-50	100	0	0
1020	天竺鼠車車	nul	60	60	1	2
1021	沙漏	nul	80	-25	1	2
1022	紅藥水	nul	250	0	0	0
1023	魔方	nul	0	800	0	0
1024	飛彈	nul	300	-50	0	-2


Cola
CoCo

第六節　特效

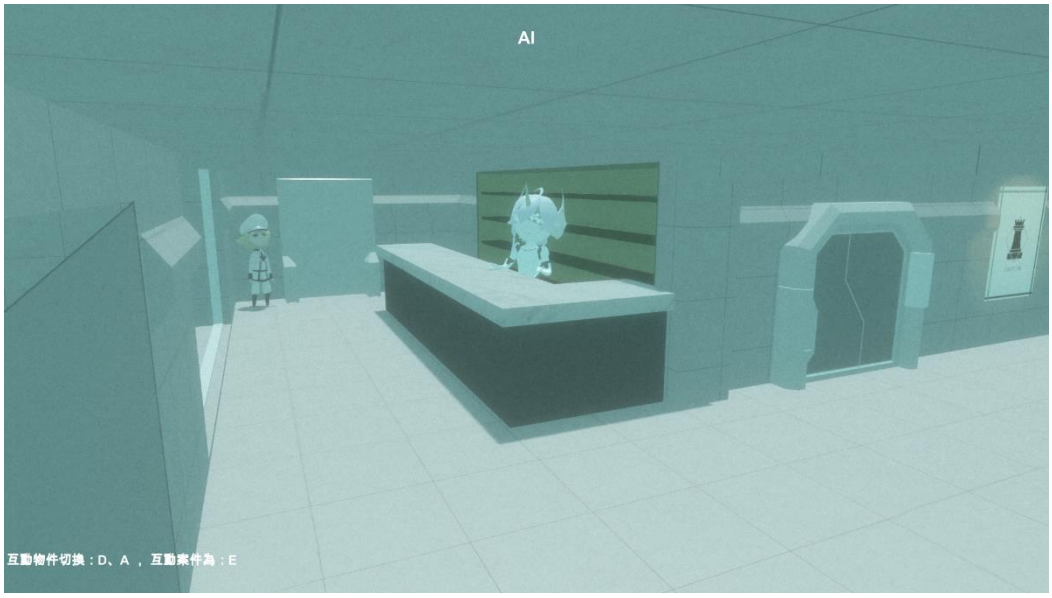
各式開槍特效

第六章 介面規劃

第一節 主選單

主畫面:

主畫面並非以傳統2D 選單製作，是以全3D 場景，並以點擊對應設施來進行互動。



設定:

遊戲設定:按鍵設定，滑鼠靈敏度。

顯示設定:視窗全螢幕切換、解析度、畫質。

音效設定:各類音效音量大小。

第二節 遊戲介面

遊戲畫面

左下為血量護甲，右下為剩餘子彈數
上面是保護目標剩餘血量，下面為經驗值

背包

左側為人物當前數值，中間為選擇裝備的數值，右上為當前任務目標
下面則是裝備欄

裝備升級器

中間可以見到升級後的數值，於右側確認升級

武器交換機

左側為本輪可換的武器列表，右側可看詳細數值並交換

第三節 對話系統

傳統文字冒險遊戲的呈現方式，人物立繪+對話內容，按下 Enter 下一頁

第七章 遊戲 AI

第一節 怪物行為列表

攻擊與移動

第二節 怪物攻擊及遊走行為

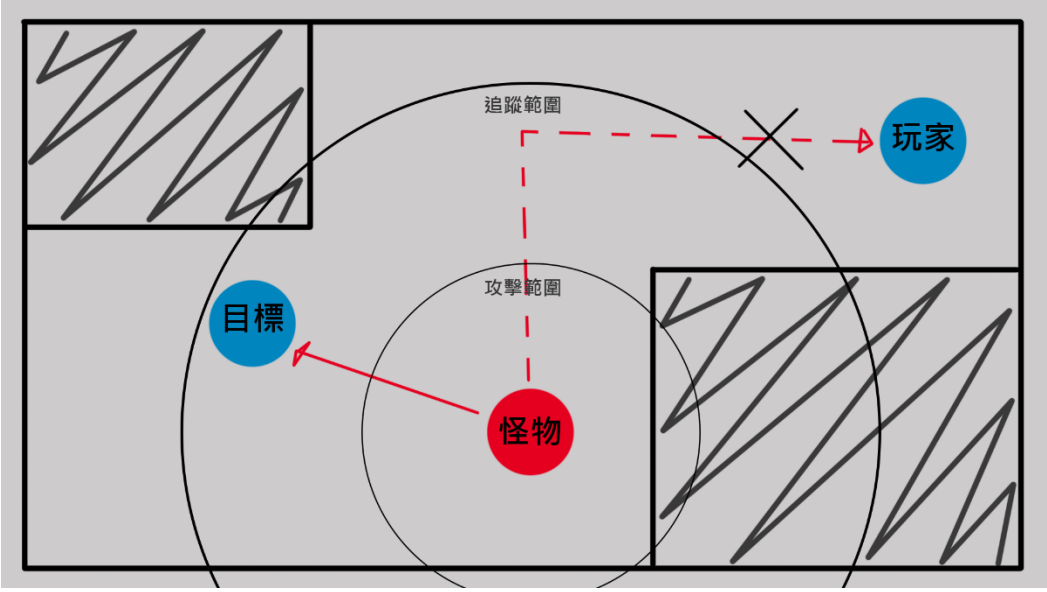
甲、移動

甲、會根據各怪物的追蹤範圍來抓目標

乙、若玩家不再範圍內，則不會追蹤，反之則會

丙、若關卡目標為守護，則每隻怪物追蹤目標則修改為最近的目標

丁、若附近有障礙物則會自動避開

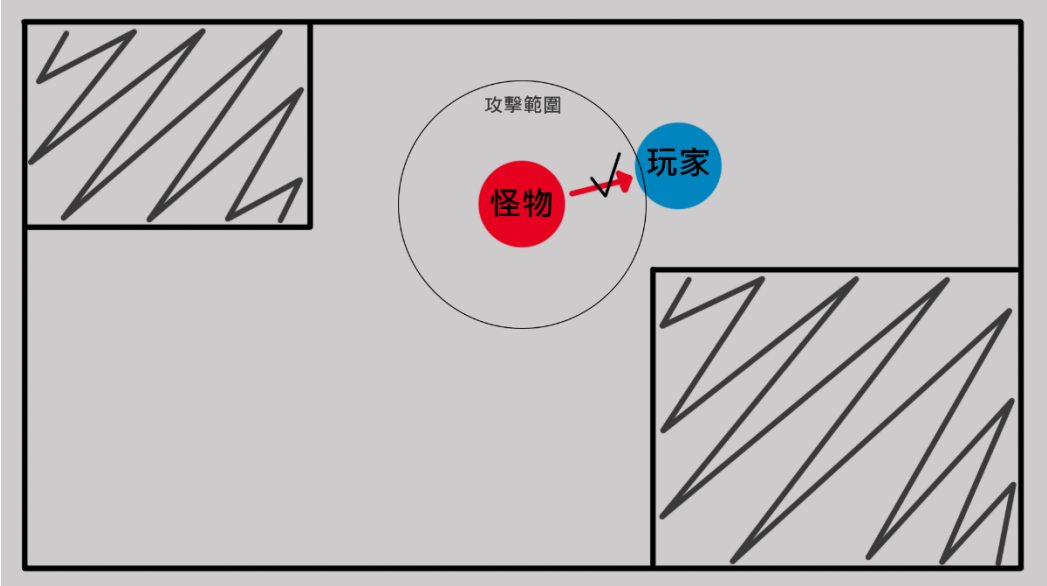


乙、攻擊

甲、若對方在追蹤範圍裡，但進入攻擊範圍怪則會停下來

乙、當進入攻擊範圍內便會開始攻擊

丙、若玩家逃離則執行移動判斷


攻擊範圍
怪物
玩家


護甲
0.00/25.00
血量
47.50/100.00
彈藥
28/30
經驗值
75.00/9375.00

第八章 遊戲系統

第一節 戰鬥系統

甲、武器

1. 武器於關卡裡的交換機取得
2. 基本素質，每把武器都會有，而每把武器的數值會有差異
   I. 傷害
   II. 射速
   III. 彈夾容量
   IV. 換彈時間
   V. 額外效果(每把武器會有專屬效果詞條)
3. 畫面上不會有準星，玩家須自行判斷中央

乙、裝備

1. 主要增強基本數值
2. 擊殺怪物可以獲得裝備
3. 裝備效果與強化數值可透過金幣升級
4. 背包中的裝備可隨時替換
5. 裝備有額外數值和被動技能，沒有主動技能

丙、戰鬥

1. 左鍵開槍，對怪物造成傷害
2. 右鍵瞄準，降低子彈飄移幅度
3. 不要與怪物接觸，將會被造成傷害

第二節 遊戲平衡

甲、各個名稱說明

1. 基礎數值:角色等級1時的基本數值
2. 成長數值:每次升等時所獲得的數值
3. 額外數值:其他因素所獲得數值，例如裝備與武器
4. 基礎倍率:1絕對不變
5. 額外倍率:因其他因素獲得的倍率，例如裝備與武器
6. 難度倍率:當前遊戲強度等級，請見下表遊戲強度計算公式
7. ATK:當前傷害
8. LV:當前等級

乙、數值計算公式

1. 玩家所有數值

   (基礎數值+成長數值+額外數值)*(基本倍率*額外倍率)=實際數值
2. 怪物所有數值

   (基礎數值+額外數值)*難度倍率*額外倍率=實際數值

```
maxHealthPoint = (100 + 0) * (level * 0.25f) * 1;
maxSheldPoint = (100 + 0) * (level * 0.25f) * 1;
speed = (15 + 0) * (level * 0.05f) * 1;
damageValue = (15 + 0) * (level * 0.25f) * 1;
attackRate = (1 + 0) * (level * 0.1f) * 1;
shieldPoint = maxSheldPoint;
healthPoint = maxHealthPoint;
nma.speed = speed;
```

丙、等級成長公式

1. 經驗值
   - I. 0升1所需經驗為 ATK*125
   - II. 之後每等級追加*(LV/8)+ATK*25
   - III. 每5級 成長數值多1倍
     - 小數點進位 最小為1
     - 基礎數值先乘等級 之後再加上 ATK*25
2. 升級成長
   - I. 血量：每等級+25*(LV/4)
   - II. 護甲：每等級+ 10*(LV/4)
   - III. 傷害：每等級+15*(LV/5)

```
maxHealthPoint = 100f + (25f * Mathf.CeilToInt(level / 4));
healthPoint = maxHealthPoint;
maxSheldPoint = 25f + (10f * Mathf.CeilToInt(level / 4));
shieldPoint = maxSheldPoint;
maxExperiencePoint = (WeaponBase.Instace.weapon.damageVlaue * 125f) + (Mathf.CeilToInt(level / 8) * WeaponBase.Instace.weapon.damageVlaue * 25f);
if (isLevelUp == true)
{
    experiencePoint = 0;
    Inventory.Instace.ItemsValueRefresh();
}
RefreshValuePoint();
```

丁、遊戲強度計算公式

遊玩難度	簡單		普通		困難	
	每經過	難度加值	每經過	難度加值	每經過	難度加值
時間	10分鐘	+1	5分鐘	+1	2分鐘	+1
玩家等級	2等級	+1	2等級	+2	1等級	+2
關卡	4關	+1	2關	+1	1關	+1
	時間 等級 關卡，每經過一段都會增加怪物難度					

第三節 背包系統

甲、查看當前數值

甲、可以看到受裝備影響後的數值，以及確認當前等級

乙、確認關卡目標

甲、可以了解當前任務是否完成，還差多少

丙、瀏覽已獲得的裝備及其效果

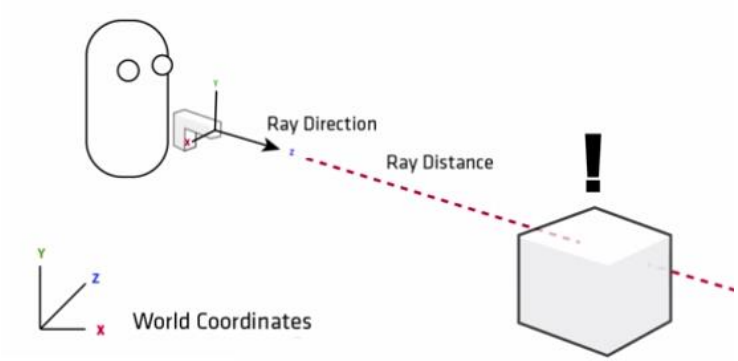
甲、查看裝備數量，裝備等級，也可拋棄不要的裝備

第四節 程式設計

1. 槍枝的系統

發射子彈：

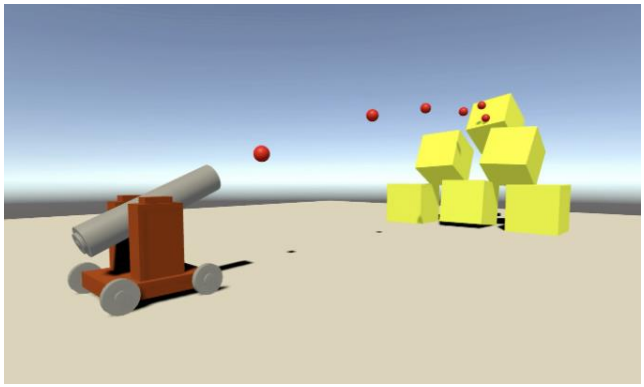
射線



在剛開始開發時，使用射線的方式讓玩家能夠擊中目標並讓目標扣血死亡，但後來發現這個方法會沒有彈道下墜的功能，除非使用數學的方式讓路線軌道彎曲符合物理，在加上使用射線會因為敵人碰撞器過小或者過遠導致沒有被射線偵測到，後來改用實例化的方式。

```
81      #region UpdateLine
        1 個參考
82      protected virtual void UpdateLine()
83      {
84          hit = CreateRaycast(5000);
85
86          if (hit.collider != null)
87          {
88              endPosition = hit.point;
89              isGetHit = true;
90          }
91          else
92          {
93              endPosition = muzzle.position + (-muzzle.forward * 5000);
94              isGetHit = false;
95          }
96      }
97      #endregion
98
99      #region RaycastHit CreateRaycast
        1 個參考
100     private RaycastHit CreateRaycast(float length)
101     {
102         Ray ray = new Ray(muzzle.position, -muzzle.forward);
103         Physics.Raycast(ray, out hit, length);
104         return hit;
105     }
106     #endregion
107
        0 Unity Message | 0 個參考
108     protected virtual void OnDrawGizmos()
109     {
110         if (isGetHit)
111         {
112             Gizmos.color = Color.red;
113             Gizmos.DrawLine(muzzle.position, endPosition);
114         }
115         else
116         {
117             Gizmos.color = Color.green;
118             Gizmos.DrawLine(muzzle.position, endPosition);
119         }
120     }
121 }
122
```

實例化

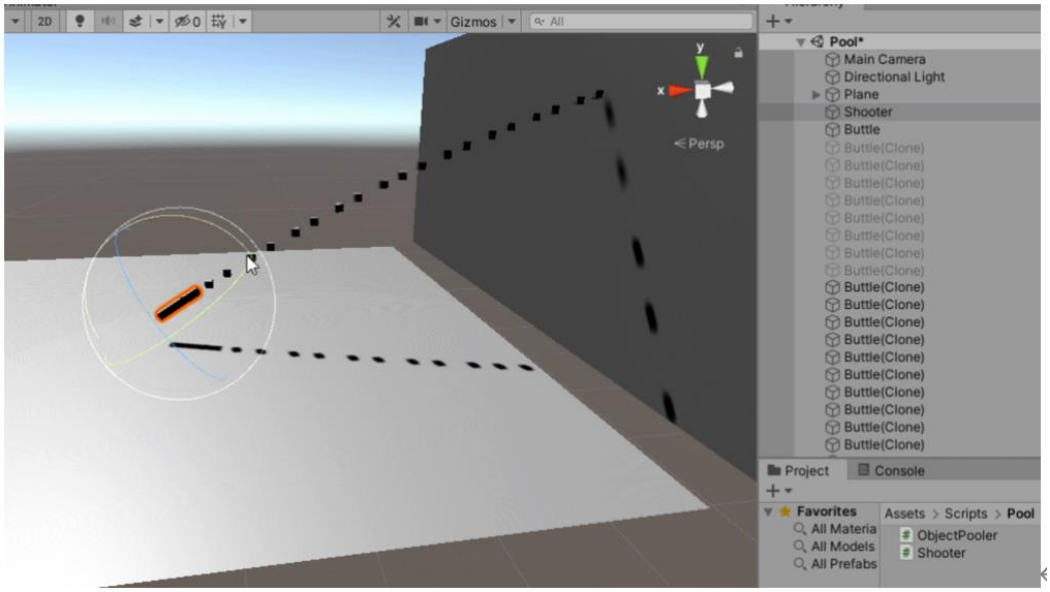


單純的就是將子彈製作成預製物件(Prefab)，再讓腳本控制實例化(Instantiate)，是十分消耗 CPU 效能的手段，在每次的實例化(Instantiate)與消除(Destroy)遊戲物件都會對玩家電腦的 CPU 造成負擔，所以得使用物件管理池。

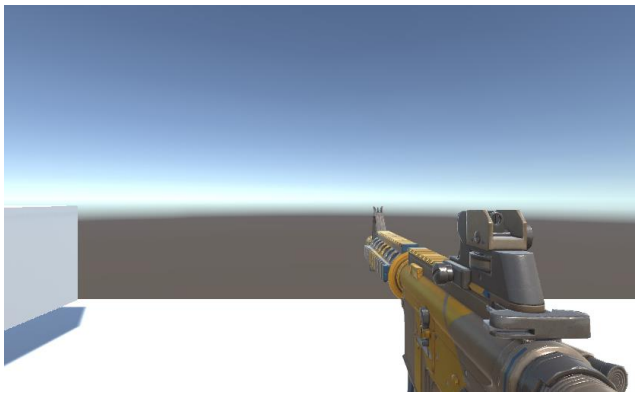
物件管理池

什麼是物件管理池?它是在遊戲一開始時，將玩家在遊戲中會使用的遊戲物件生成在遊戲場景中並隱藏起來，在玩家需要使用時他會從池中撈出閒置的遊戲物件來使用，當玩家使用完畢時不會直接銷毀，而是將是再度隱藏收回池中等到下一次再拿出來使用，這樣就能大大減少 CPU 的負擔。

```
    Unity Message | 0 個參考
28  private void Start()
29  {
30      poolDictionary = new Dictionary<string, Queue<GameObject>>();
31
32      foreach (Pool pool in pools)
33      {
34          Queue<GameObject> objectPool = new Queue<GameObject>();
35
36          for (int i = 0; i < pool.size; i++)
37          {
38              GameObject obj = Instantiate(pool.prefab);
39              obj.SetActive(false);
40              objectPool.Enqueue(obj);
41          }
42          poolDictionary.Add(pool.tag, objectPool);
43      }
44  }
45
    1 個參考
46  public GameObject SpawnFromPool(string tag,Vector3 pos,Quaternion rot)
47  {
48      if (!poolDictionary.ContainsKey(tag))
49      {
50          Debug.LogWarning("Pool with tag" + tag + "doesn't excist");
51          return null;
52      }
53
54      GameObject objectToSpawn = poolDictionary[tag].Dequeue();
55
56      objectToSpawn.SetActive(true);
57      objectToSpawn.transform.position = pos;
58      objectToSpawn.transform.rotation = rot;
59
60      poolDictionary[tag].Enqueue(objectToSpawn);
61
62      return objectToSpawn;
63  }
64  }
65
```

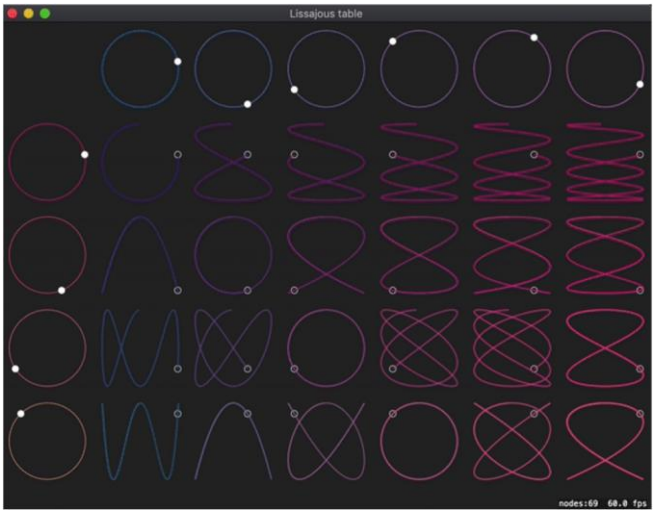


持槍的搖擺：



上圖可以看到在持槍時，缺少了呼吸或者武器搖擺的效果感覺很僵硬，所以我們使用利薩茹曲線(Lissajous Curve)，可以完美的循環，並透過修改參數有不同的形狀。

```
28      private void AimDownSightUpdate()
29      {
30          current.localPosition = m_swayPos;
31      }
32
33      private void UpdateSwayOffset()
34      {
35          Vector3 targetPos = (LissajousCurve(m_theta, m_A, Mathf.PI, m_B) /
36              m_sizeReducerFactor);
37          m_swayPos = Vector3.Lerp(m_swayPos, targetPos,
38              Time.smoothDeltaTime * m_swayLerpSpeed);
39          m_theta += m_thetaIncreaseFactor;
40      }
41
42      private Vector3 LissajousCurve(float theta, float A, float delta, float B)
43      {
44          Vector3 pos = Vector3.zero;
45          pos.x = Mathf.Sin(theta);
46          pos.y = A * Mathf.Sin(B * theta + delta);
47          return pos;
48      }
49  }
```



後座力晃動：

槍枝在射擊時一定有的後座力，它會影響玩家的子彈走向以及給予玩家射擊回饋，並且可以透過參數來設定後座力強度配合設計不同的槍枝。

```
    @Unity Message|0 個參考
    void FixedUpdate()
    {
        CurrentRecoil1 = Vector3.Lerp(CurrentRecoil1, Vector3.zero, Recoil1 * Time.deltaTime);
        CurrentRecoil2 = Vector3.Lerp(CurrentRecoil2, CurrentRecoil1, Recoil2 * Time.deltaTime);
        CurrentRecoil3 = Vector3.Lerp(CurrentRecoil3, Vector3.zero, Recoil3 * Time.deltaTime);
        CurrentRecoil4 = Vector3.Lerp(CurrentRecoil4, CurrentRecoil3, Recoil4 * Time.deltaTime);

        RecoilPositionTranform.localPosition = Vector3.Slerp(RecoilPositionTranform.localPosition, CurrentRecoil3, PositionDampTime * Time.fixedDeltaTime);
        RotationOutput = Vector3.Slerp(RotationOutput, CurrentRecoil1, RotationDampTime * Time.fixedDeltaTime);
        RecoilRotationTranform.localRotation = Quaternion.Euler(RotationOutput);
    }
    1 個參考
    public void Fire()
    {
        if (aim == true)
        {
            CurrentRecoil1 += new Vector3(RecoilRotation_Aim.x, Random.Range(-RecoilRotation_Aim.y, RecoilRotation_Aim.y), Random.Range(-RecoilRotation_Aim.z, RecoilRotation_Aim.z));
            CurrentRecoil3 += new Vector3(Random.Range(-RecoilKickBack_Aim.x, RecoilKickBack_Aim.x), Random.Range(-RecoilKickBack_Aim.y, RecoilKickBack_Aim.y), RecoilKickBack_Aim.z);
        }
        if (aim == false)
        {
            CurrentRecoil1 += new Vector3(RecoilRotation.x, Random.Range(-RecoilRotation.y, RecoilRotation.y), Random.Range(-RecoilRotation.z, RecoilRotation.z));
            CurrentRecoil3 += new Vector3(Random.Range(-RecoilKickBack.x, RecoilKickBack.x), Random.Range(-RecoilKickBack.y, RecoilKickBack.y), RecoilKickBack.z);
        }
    }
}
```

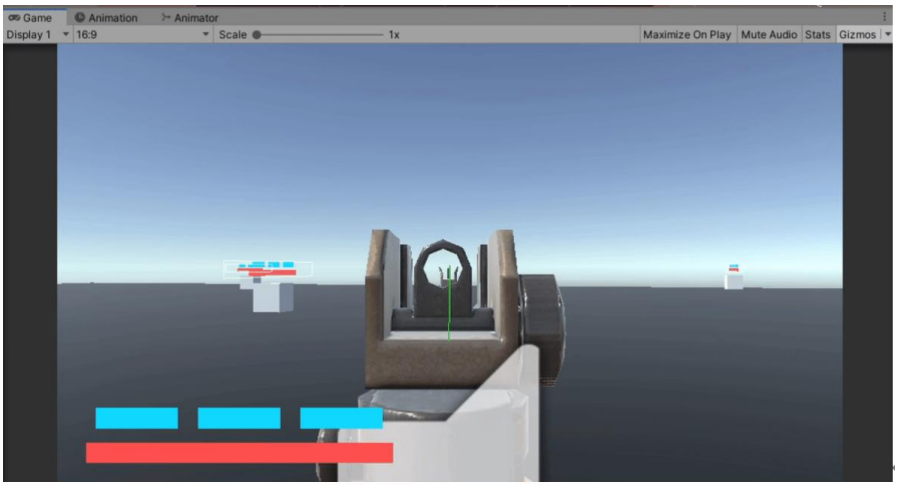
瞄準與腰部射擊狀態：

在瞄準時，後座力會降低並且不會有子彈擴散的問題；在腰射時，後座力會增加並且有子彈擴散效果，可以透過各個參數設定擴散範圍。

```
    void Shoot()
    {
        nextTimeToFire = Time.time + 1f / fireRate;
        recoil.Fire();
        factor = Mathf.Lerp(factor, 5f, Time.deltaTime * factorSpeed);

        if(Physics.Raycast(fpsCam.transform.position, fpsCam.transform.forward, out hit, range))
        {
            isGetHit = true;
            if (!isAiming)
            {
                x = Random.Range(maxDiffusion.x, -maxDiffusion.x) * factor;
                y = Random.Range(maxDiffusion.y, -maxDiffusion.y) * factor;
                muzzle.localRotation = Quaternion.Euler(x, y, 0);
            }
            else
            {
                muzzle.localRotation = Quaternion.identity;
            }
            Debug.Log(hit.transform.name);
        }
        else
        {
            isGetHit = false;
        }
    }
```

```
    void Aim()
    {
        if (Input.GetKey(KeyCode.Mouse1))
        {
            current.localPosition = Vector3.Slerp(current.localPosition, aimPosition, 2.5f * Time.fixedDeltaTime);
            isAiming = true;
            recoil.aim = true;
        }
        else
        {
            current.localPosition = Vector3.Slerp(current.localPosition, generalPosition, 2.5f * Time.fixedDeltaTime);
            isAiming = false;
            recoil.aim = false;
        }
    }
```

後期也增加了，瞄準時會有聚焦效果。



填裝子彈：

在初期製作填裝子彈只有數值上的補充，在視覺上是沒有的，因為槍枝種類多樣，要製作大量素材動畫是費時的，所以就使用程式控制讓槍向下移動當作是換彈動畫，這樣可以適用在各種槍枝種類也能透過裝備與玩家數值影響換彈時間來增加遊戲性。

```
182     protected virtual IEnumerator Reloading()
183     {
184         if (Input.GetKeyDown(InputManagers.Instace.reloadKey) && !isReloading)
185         {
186             Sound(weapon.reloadSound);
187             isReloading = true;
188             yield return new WaitForSeconds(weapon.reloadTime);
189             currentAmmo = weapon.maxAmmo;
190             isReloading = false;
191         }
192         if (isReloading)
193         {
194             recoil_Center.localPosition = Vector3.Slerp(recoil_Center.localPosition, new Vector3(0, -0.5f, 0), weapon.reloadTime * Time.fixedDeltaTime);
195         }
196     }
```

武器的數值與素材：

因為考慮未來武器的數量眾多，為了讓企劃與美術方便在設定武器數值與美術素材的導入，則使用 Unity 讀取 Excel 資料來抓取編輯好的數值與美術素材的名稱，並生成整理武器的資料導入遊戲中。

首先先抓取資源位置。

```
66      1 個參考
67      public void StartLoadWeaponData()
68      {
69          TextAsset WeaponsData = Resources.Load<TextAsset>("Excel/WeaponData");
70          weaponSprites = Resources.LoadAll("Weapon/WeaponSprites", typeof(Sprite)).Cast<Sprite>().ToArray();
71          WeaponMeshs = Resources.LoadAll("Weapon/WeaponMeshs", typeof(Mesh)).Cast<Mesh>().ToArray();
72          weaponMaterials = Resources.LoadAll("Weapon/WeaponMaterials", typeof(Material)).Cast<Material>().ToArray();
73          weaponObject = Resources.LoadAll("Weapon/WeaponObjects", typeof(GameObject)).Cast<GameObject>().ToArray();
74          weaponShootSound = Resources.LoadAll("Weapon/WeaponShootSounds", typeof(AudioClip)).Cast<AudioClip>().ToArray();
75          weaponReloadSound = Resources.LoadAll("Weapon/WeaponReloadSounds", typeof(AudioClip)).Cast<AudioClip>().ToArray();
```

先讀取武器有幾把，再將數量用迴圈每次執行一次新增一把武器資料，抓取 Excel 的資料數值導入給它。

```
string[] weaponData = WeaponsData.text.Split(new char[] { '\n' });

for (int i = 1; i < weaponData.Length - 1; i++)
{
    string[] weaponRow = weaponData[i].Split(new char[] { ',' });

    if (weaponData[1] != " ")
    {
        Weapon weapon = new Weapon();
        weapon.name = weaponRow[1];
        int.TryParse(weaponRow[0], out weapon.ID);
        weapon.itemName_ch = weaponRow[1];
        weapon.itemName_en = weaponRow[2];
        float.TryParse(weaponRow[3], out weapon.damageVlaue);
        float.TryParse(weaponRow[4], out weapon.fireRate);
        int.TryParse(weaponRow[5], out weapon.maxAmmo);
        float.TryParse(weaponRow[6], out weapon.factorSpeed);
        float.TryParse(weaponRow[7], out weapon.aimTime);
        float.TryParse(weaponRow[8], out weapon.reloadTime);
        float.TryParse(weaponRow[9], out weapon.maxDiffusion.x);
        float.TryParse(weaponRow[10], out weapon.maxDiffusion.y);
        float.TryParse(weaponRow[11], out weapon.aimPosition.x);
        float.TryParse(weaponRow[12], out weapon.aimPosition.y);
        float.TryParse(weaponRow[13], out weapon.aimPosition.z);
        float.TryParse(weaponRow[14], out weapon.aimRotation.x);
        float.TryParse(weaponRow[15], out weapon.aimRotation.y);
        float.TryParse(weaponRow[16], out weapon.aimRotation.z);
        float.TryParse(weaponRow[17], out weapon.generalPosition.x);
        float.TryParse(weaponRow[18], out weapon.generalPosition.y);
        float.TryParse(weaponRow[19], out weapon.generalPosition.z);
        float.TryParse(weaponRow[20], out weapon.generalRotation.x);
        float.TryParse(weaponRow[21], out weapon.generalRotation.y);
        float.TryParse(weaponRow[22], out weapon.generalRotation.z);
        float.TryParse(weaponRow[23], out weapon.meshPosition.x);
        float.TryParse(weaponRow[24], out weapon.meshPosition.y);
        float.TryParse(weaponRow[25], out weapon.meshPosition.z);
        float.TryParse(weaponRow[26], out weapon.meshRotation.x);
        float.TryParse(weaponRow[27], out weapon.meshRotation.y);
        float.TryParse(weaponRow[28], out weapon.meshRotation.z);
        float.TryParse(weaponRow[29], out weapon.muzzlePosition.x);
        float.TryParse(weaponRow[30], out weapon.muzzlePosition.y);
        float.TryParse(weaponRow[31], out weapon.muzzlePosition.z);
        float.TryParse(weaponRow[32], out weapon.m_A);
        float.TryParse(weaponRow[33], out weapon.m_B);
        float.TryParse(weaponRow[34], out weapon.m_sizeReducerFactor);
```

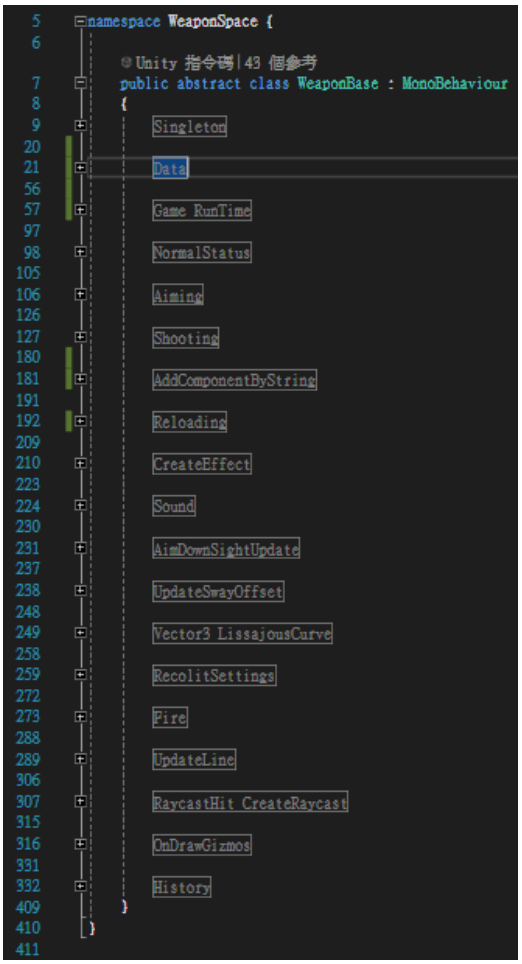
再將 Unity 的美術素材導入給它，加入清單(List)。

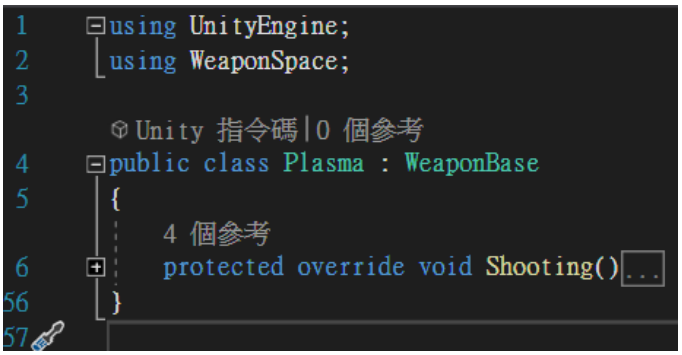
```
        weapon.introduction_ch = weaponRow[59];
        weapon.introduction_en = weaponRow[60];
        for (int s = 0; s < weaponSprites.Length; s++)
        {
            if (weaponSprites[s].name == weaponRow[2])
            {
                weapon.icon = weaponSprites[s];
            }
        }

        for (int ms = 0; ms < WeaponMeshs.Length; ms++)
        {
            if (WeaponMeshs[ms].name == weaponRow[2])
            {
                weapon.mesh = WeaponMeshs[ms];
            }
        }

        for (int ml = 0; ml < weaponMaterials.Length; ml++)
        {
            if (weaponMaterials[ml].name == weaponRow[2])
            {
                weapon.material = weaponMaterials[ml];
            }
        }

        for (int os = 0; os < weaponObject.Length; os++)
        {
            if (weaponObject[os].name == weaponRow[2])
            {
                weapon.weaponObject = weaponObject[os];
            }
        }

        for (int wss = 0; wss < weaponShootSound.Length; wss++)
        {
            if (weaponShootSound[wss].name == weaponRow[57])
            {
                weapon.shootSound = weaponShootSound[wss];
            }
        }

        for (int wrs = 0; wrs < weaponReloadSound.Length; wrs++)
        {
            if (weaponReloadSound[wrs].name == weaponRow[58])
            {
                weapon.reloadSound = weaponReloadSound[wrs];
            }
        }
        weapons.Add(weapon);
```

這樣就能夠快速又簡潔的將武器新增到遊戲中，並且在未來能夠快速的調整數值與變換美術資源。

結論與展示：

綜合上述的功能組合再一起為基類(Base)，再利用繼承(Inheritance)與覆寫(Override)的方式，就能製作許多不同種類的武器功能。





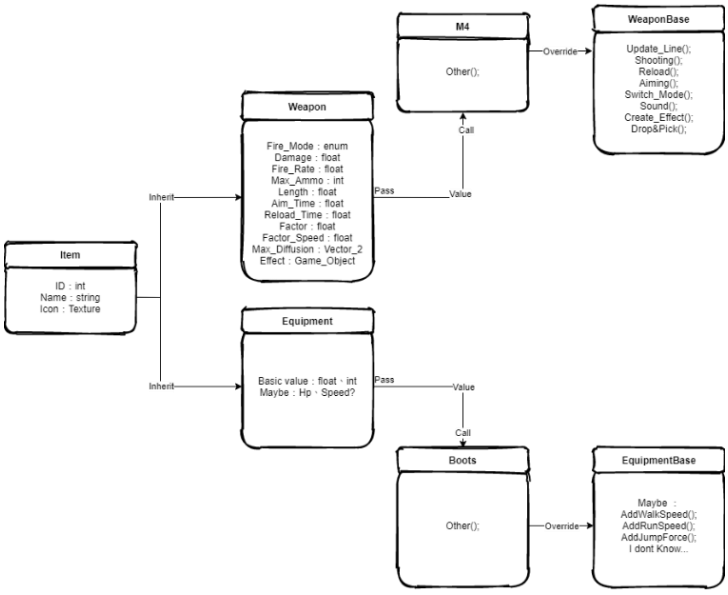
2.裝備與武器的分類與使用

建立基類與資料數據：

在遊戲中有裝備也有武器他們同樣都為遊戲中的物品(Item)都有 ID 與名稱以及介紹……等，重點是他們是大量的，為了方便儲存與讀取大量的資料就需要使用 Scriptable Object，這樣一來就不需要製作大量的預製物件。

```
using UnityEngine;
Unity 指令碼|6 個參考
public abstract class Item : ScriptableObject
{
    [Header("身分")]
    public int ID;
    public string itemName_ch;
    public string itemName_en;
    public Sprite icon;
    [Header("介紹")]
    [Space(15)]
    [TextArea(15, 15)]
    public string introduction_ch;
    [TextArea(15, 15)]
    public string introduction_en;

    public Mesh mesh;
    public Material material;
}
```

繼承與分類：



如上述所說，遊戲中的物件分為武器與裝備，他們的重複的資料內容很多，也有不重複的資料，必須將他們區隔開，所以就需要使用繼承(Inheritance)的方式，繼承 Item 分類為 Weapon 與 Equipment。

```
using UnityEngine;

[System.Serializable]
[CreateAssetMenu(fileName = "New Equipment", menuName = "Create/Equipment")]
Unity 指令碼|16 個參考
public sealed class Equipment : Item
{
    [Header("物件")]
    [Space(15)]
    public GameObject prefab;
    [Header("裝備效果")]
    [Space(15)]
    public int level;
    public int addHealthValue;
    public int addSheldValue;
    public float addWalkSpeedValue;
    public float addRunSpeedValue;
    public float addReloadSpeed;
    public float addAimSpeed;
    public float addJumpForce;
    [Header("裝備特效")]
    [Space(15)]
    public ParticleSystem equipmentEffect;

}
```

```
using UnityEngine;

[System.Serializable]
[CreateAssetMenu(fileName = "New Weapon", menuName = "Create/Weapon")]
Unity 指令碼|13 個參考
public class Weapon : Item
{
    [Header("武器數值")]
    [Space(15)]
    public float damageVlaue = 10f;
    public float fireRate = 3f;
    public int maxAmmo = 30;
    public float shootLength = 5000f;
    public float factorSpeed = 1.5f;
    public float aimTime = 4f;
    public float reloadTime = 2.5f;
    public Vector2 maxDiffusion = new Vector2(1.5f, 1.5f);
    public bool Safe = true, Semi, Burst, Auto;

    [Header("組件位置")]
    [Space(15)]
    public Vector3 aimPosition;
    public Vector3 aimRotation;
    public Vector3 generalPosition;
    public Vector3 generalRotation;
    public Vector3 meshPosition;
    public Vector3 meshRotation;
    public Vector3 muzzlePosition;
```

讀取並導入資料生成物件：

透過讀取事先填寫好的 Excel，導入對象資料，透過 ID 的比對轉換成對應物件。

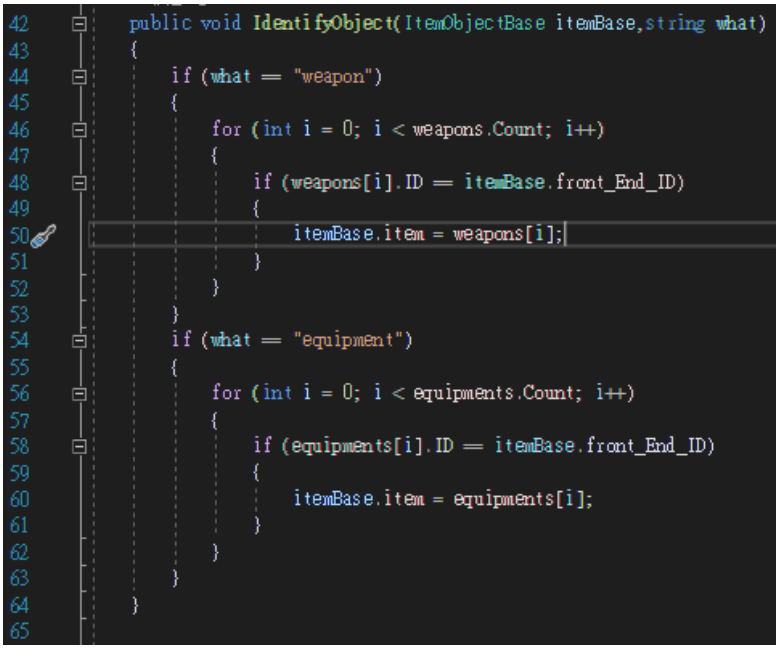
先是讀取 Excel 資料與美術素材路徑，新增一個對象來導入資料。

```
public void StartLoadEquipmentData()
{
    TextAsset equipmentsData = Resources.Load<TextAsset>("Excel/EquipmentData");
    equipmentsSprites = Resources.LoadAll("Item/ItemSprites", typeof(Sprite)).Cast<Sprite>().ToArray();
    equipmentsMeshs = Resources.LoadAll("Item/ItemMeshs", typeof(Mesh)).Cast<Mesh>().ToArray();
    equipmentsMaterials = Resources.LoadAll("Item/ItemMaterials", typeof(Material)).Cast<Material>().ToArray();

    string[] equipmentData = equipmentsData.text.Split(new char[] { '\n' });

    for (int i = 1; i < equipmentData.Length - 1; i++)
    {
        string[] equipmentRow = equipmentData[i].Split(new char[] { ',' });

        if (equipmentRow[1] != "")
        {
            Equipment equipment = new Equipment();
            equipment.name = equipmentRow[1];
            int.TryParse(equipmentRow[0], out equipment.ID);
            equipment.itemName_ch = equipmentRow[1];
            equipment.itemName_en = equipmentRow[2];
            int.TryParse(equipmentRow[3], out equipment.addHealthValue);
            int.TryParse(equipmentRow[4], out equipment.addSheldValue);
            float.TryParse(equipmentRow[5], out equipment.addWalkSpeedValue);
            float.TryParse(equipmentRow[6], out equipment.addRunSpeedValue);
            float.TryParse(equipmentRow[7], out equipment.addReloadSpeed);
            float.TryParse(equipmentRow[8], out equipment.addAimSpeed);
            float.TryParse(equipmentRow[9], out equipment.addJumpForce);
            equipment.introduction_ch = equipmentRow[14];
            equipment.introduction_en = equipmentRow[15];
```

抓取相同名稱的美術素材，導入對象後加入清單(List)。

```
            for (int s = 0; s < equipmentsSprites.Length; s++)
            {
                if (equipmentsSprites[s].name == equipmentRow[1])
                {
                    equipment.icon = equipmentsSprites[s];
                }
            }

            for (int ms = 0; ms < equipmentsMeshs.Length; ms++)
            {
                if (equipmentsMeshs[ms].name == equipmentRow[1])
                {
                    equipment.mesh = equipmentsMeshs[ms];
                }
            }

            for (int ml = 0; ml < equipmentsMaterials.Length; ml++)
            {
                if (equipmentsMaterials[ml].name == equipmentRow[1])
                {
                    equipment.material = equipmentsMaterials[ml];
                }
            }
            equipments.Add(equipment);
        }
    }
```

製作一個方法透過迴圈找到相同 ID，導入遊戲資源，這樣只需要一個基類的遊戲物件來執行這些事情，就不需要因為有大量的裝備製作大量的預製物件。



```
42      public void IdentifyObject(ItemObjectBase itemBase,string what)
43      {
44          if (what == "weapon")
45          {
46              for (int i = 0; i < weapons.Count; i++)
47              {
48                  if (weapons[i].ID == itemBase.front_End_ID)
49                  {
50                      itemBase.item = weapons[i];
51                  }
52              }
53          }
54          if (what == "equipment")
55          {
56              for (int i = 0; i < equipments.Count; i++)
57              {
58                  if (equipments[i].ID == itemBase.front_End_ID)
59                  {
60                      itemBase.item = equipments[i];
61                  }
62              }
63          }
64      }
65
```

3.NPC 對話系統

實作：

為了製作讓遊戲中玩家能夠與 NPC 對話，在大量的 NPC 中獲取對應 NPC 的隨機大量訊息，所以製作讀取 Excel 的訊息轉換成資料加入清單(List)，從清單中隨機抽取訊息給與玩家回覆。

	A	B	C	D	E	F	G	H	I
1	ID	名稱	Name						
2	1001	米勒	Miller	怎麼了!新	抱歉…口	有甚麼事	你知道這	E N D.	
3	1002	米勒	Miller	注意!!!!!!!	朋友!	喜歡	E N D.		
4	1003	米勒	Miller	我...還在	嗯?你怎麼	不要理我!	E N D.		
5	1004	米勒	Miller	告訴你!	我	E N D.			
6	1005	米勒	Miller	不知道弟	E N D.				
7	1006	米勒	Miller	弟弟不知	你怎麼在	去旁邊!!!!	E N D.		
8	1007	米勒	Miller	看到我不	在這裡幹	E N D.			

讀取 Excel 資料與 NPC 的美術圖，因為每一個訊息字句都不一定，所以透過辨識此句子是否為 E N D. 或者 空白 並設為斷句，如此一來就不用固定句數。

```
public void StartLoadData()
{
    TextAsset messageData = Resources.Load<TextAsset>("Excel/MessageData");
    npcSprites = Resources.LoadAll("NPC/Sprite", typeof(Sprite)).Cast<Sprite>().ToArray();
    npc = FindObjectsOfType<NPCBase>();
    string[] data = messageData.text.Split(new char[] { '\n' });

    for (int i = 1; i < data.Length - 1; i++)
    {
        string[] row = data[i].Split(new char[] { ',' });

        if (row[1] != "")
        {
            Message message = new Message();
            int.TryParse(row[0], out message.messageID);
            message.name = row[0];
            message.npcName_Ch = row[1];
            message.npcName_En = row[2];
            List<string> tmp = new List<string>();
            for (int r = 3; r < row.Length; r++)
            {
                if (row[r].Contains("E N D.") || row[r] == "" || row[r] == null)
                    break;
                tmp.Add(row[r]);
                message.message = tmp.ToArray();
            }
            messages.Add(message);
        }
    }
}
```

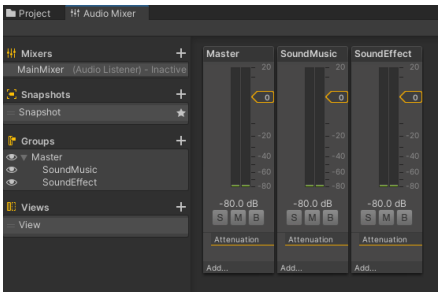
辨識 NPC 名稱，導入資料訊息與圖片。

```
public void CheckNPCName()
{
    for (int i = 0; i < npc.Length; i++)
    {
        for (int j = 0; j < messages.Count; j++)
        {
            if (npc[i].npcName == messages[j].npcName_En)
            {
                npc[i].randomMessage.Add(messages[j]);
            }
        }
        for (int p = 0; p < npcSprites.Length; p++)
        {
            if (npc[i].npcName == npcSprites[p].name)
            {
                npc[i].photo = npcSprites[p];
            }
        }
    }
}
```

4. 遊戲設定系統

遊戲基本設定：

使用 Audio Mixer 控制主音量、音效、音樂。




```
#region Slider
2 個參考
public void SetMainSoundVolume(float volume)
{
    audioMixer.SetFloat("MainSound", Mathf.Lerp(-80, 20, volume / 100));
}

2 個參考
public void SetSoundEffectVolume(float volume)
{
    audioMixer.SetFloat("SoundEffect", Mathf.Lerp(-80, 20, volume / 100));
}

2 個參考
public void SetSoundMusicVolume(float volume)
{
    audioMixer.SetFloat("SoundMusic", Mathf.Lerp(-80, 20, volume / 100));
}
#endregion
```

利用 Unity 給的 API 控制遊戲品質與遊戲視窗模式。

```
#region DropDown
2 個參考
public void SetQuality(int qualityIndex)
{
    QualitySettings.SetQualityLevel(qualityIndex);
}

2 個參考
public void SetResolutions(int resolutionIndex)
{
    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
}
#endregion

#region Toggle
2 個參考
public void SetFullScreen(bool isFullscreen)
{
    Screen.fullScreen = isFullscreen;
}
#endregion
```

遊戲熱鍵設定：

先建立玩家的熱鍵對象與資料。

```
Unity 指令碼 | 99+ 個參考
public class InputManagers : MonoBehaviour
{
    Singleton

    public string horizontalInputName = "Horizontal", verticalInputName = "Vertical";
    public string mouseXInputName = "Mouse X", mouseYInputName = "Mouse Y";
    public float mouseSensitivity_Normal = 150f;
    public float mouseSensitivity_Aim = 150f;
    public KeyCode frontKey;
    public KeyCode backKey;
    public KeyCode leftKey;
    public KeyCode rightKey;
    public KeyCode runKey;
    public KeyCode jumpKey;
    public KeyCode shootKey;
    public KeyCode aimKey;
    public KeyCode reloadKey;
    public KeyCode switchModeKey;
    public KeyCode pickUpKey;
    public KeyCode settingKey;
    public KeyCode equipmentKey;
```

當按鈕被按下時，替換對象中熱鍵資料在將資料打印在按鈕的文字。

```
void HotKeyButton()
{
    for (int i = 0; i < HotKeys.Length; i++)
    {
        if (HotKeys[i].name == "FrontKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.frontKey.ToString();
        if (HotKeys[i].name == "BackKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.backKey.ToString();
        if (HotKeys[i].name == "RightKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.rightKey.ToString();
        if (HotKeys[i].name == "LeftKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.leftKey.ToString();
        if (HotKeys[i].name == "RunKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.runKey.ToString();
        if (HotKeys[i].name == "JumpKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.jumpKey.ToString();
        if (HotKeys[i].name == "ShootKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.shootKey.ToString();
        if (HotKeys[i].name == "AimKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.aimKey.ToString();
        if (HotKeys[i].name == "ReloadKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.reloadKey.ToString();
        if (HotKeys[i].name == "SwitchModeKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.switchModeKey.ToString();
        if (HotKeys[i].name == "PickUpKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.pickUpKey.ToString();
        if (HotKeys[i].name == "SettingKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.settingKey.ToString();
        if (HotKeys[i].name == "EquipmentKeyButton")
            HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.equipmentKey.ToString();
        //if (HotKeys[i].name == "MissionKeyButton")
        //    HotKeys[i].GetComponentInChildren<Text>().text = InputManagers.Instace.missionKey.ToString();
    }
}

Unity Message | 0 個參考
private void OnGUI()
{
    keyEvent = Event.current;
    if (keyEvent.isKey && waitingForKey)
    {
        newKey = keyEvent.keyCode;
        waitingForKey = false;
    }
}
```

```
public void StartAssignment(string keyName)
{
    if (!waitingForKey)
        StartCoroutine(AssignKey(keyName));
}

0 個參考
public void SentText(Text text)
{
    buttonText = text;
}

1 個參考
IEnumerator WaitForKey()
{
    while (!keyEvent.isKey)
        yield return null;
}

1 個參考
IEnumerator AssignKey(string keyName)
{
    waitingForKey = true;
    yield return WaitForKey();
    switch (keyName)
    {
        case "FrontKey":
            InputManagers.Instace.frontKey = newKey;
            buttonText.text = InputManagers.Instace.frontKey.ToString();
            break;
        case "BackKey":
            InputManagers.Instace.backKey = newKey;
            buttonText.text = InputManagers.Instace.backKey.ToString();
            break;
        case "RightKey":
            InputManagers.Instace.rightKey = newKey;
            buttonText.text = InputManagers.Instace.rightKey.ToString();
            break;
        case "LeftKey":
```

遊戲設定後的提示：

擔心玩家設定結束後沒有儲存，所以增加一個提示，如果讀取存檔資料與玩家當前設定資料不一至，就會跳出是否儲存的訊息框提醒玩家儲存或者不儲存。

```
if (Input.GetKeyDown(InputManagers.Instace.settingKey))
{
    if (setting_Canvas.alpha == 1)
    {
        SettingsData data = SaveSystem.LoadSettings(SettingsManager.Instace);
        if (SettingsManager.Instace.soundSliders[0].value != data.mainSoundValue || SettingsManager.Instace.soundSliders[1].value != data.soundMusicValue || SettingsManager.Instace.soundSliders[2].value != data.soundEffectValue || SettingsManager.Instace.qualityDropDown.value != data.qua
        {
            SettingsManager.Instace.SureOrNo.SetActive(true);
        }
        else
        {
            CloseInterface(setting_Canvas);
            playerInterface_Canvas.alpha = 1f;
            LockCursor();
        }
        return;
    }
```

5.存檔與讀檔系統

在製作存檔與讀檔機制可以使用 PlayerPrefs 的 API 或者使用 System.IO 建立文字資料儲存與讀取資料，我們樂見玩家能夠透過外部修改遊戲資料，所以採用 System.IO 的方式製作存讀系統並分為設定、熱鍵、玩家，三種資料儲存。

玩家：

辨識玩家是否第一次開啟遊戲，如果是第一次會進入新手教學，反之讀取到大廳首頁的場景。

```
#region PlayerData
public static bool havePlayerData;

2 個參考
public static void SaveData(PlayerDataUnityFile playerDataUnityFile)
{
    BinaryFormatter formatter = new BinaryFormatter();
    string path = Path.Combine(Application.persistentDataPath + "/PlayerData.PunCry");
    FileStream stream = new FileStream(path, FileMode.Create);

    PlayerData data = new PlayerData(playerDataUnityFile);

    formatter.Serialize(stream, data);
    stream.Close();
}

1 個參考
public static PlayerData LoadData(PlayerDataUnityFile playerDataUnityFile)
{
    string path = Path.Combine(Application.persistentDataPath + "/PlayerData.PunCry");
    if (File.Exists(path))
    {
        havePlayerData = true;
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        PlayerData data = formatter.Deserialize(stream) as PlayerData;
        stream.Close();
        return data;
    }
    else
    {
        havePlayerData = false;
        SaveData(playerDataUnityFile);
        //Debug.LogError("Save file not found in " + path);
        return null;
    }
}
#endregion
```

設定：

儲存玩家基礎設定，如：音量、畫質、品質……等。

```
#region SettingsData
public static bool haveSettingsFile;

3 個參考
public static void SaveSettings(SettingsManager settings)
{
    BinaryFormatter formatter = new BinaryFormatter();
    string path = Path.Combine(Application.persistentDataPath + "/Settings.PunCry");
    FileStream stream = new FileStream(path, FileMode.Create);

    SettingsData data = new SettingsData(settings);

    formatter.Serialize(stream, data);
    stream.Close();
}

3 個參考
public static SettingsData LoadSettings(SettingsManager settings)
{
    string path = Path.Combine(Application.persistentDataPath + "/Settings.PunCry");
    if (File.Exists(path))
    {
        haveSettingsFile = true;
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        SettingsData data = formatter.Deserialize(stream) as SettingsData;
        stream.Close();
        return data;
    }
    else
    {
        haveSettingsFile = false;
        //SaveSettings(settings);
        //Debug.LogError("Save file not found in " + path);
        return null;
    }
}
#endregion
```

熱鍵：

儲存玩家按鍵設定。

```
#region HotKeyData
public static bool haveHotKeyFile;

1 個參考
public static void SaveHotKey(InputManagers inputManagers)
{
    BinaryFormatter formatter = new BinaryFormatter();
    string path = Path.Combine(Application.persistentDataPath + "/HotKeys.PunCry");
    FileStream stream = new FileStream(path, FileMode.Create);

    HotKeyData data = new HotKeyData(inputManagers);

    formatter.Serialize(stream, data);
    stream.Close();
}


public static HotKeyData LoadHotKey(InputManagers inputManagers)
{
    string path = Path.Combine(Application.persistentDataPath + "/HotKeys.PunCry");
    if (File.Exists(path))
    {
        haveHotKeyFile = true;
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        HotKeyData data = formatter.Deserialize(stream) as HotKeyData;
        stream.Close();
        return data;
    }
    else
    {
        haveHotKeyFile = false;
        //SaveHotKey(inputManagers);
        return null;
    }
}
#endregion
}
```

第九章 結論與展望

第一節 結論

經過這次開發遊戲，可得到以下之結論：
1.開發專案前必須好好評估團隊技術力，在以此來設計專案規模，避免發生某項目無法達成的窘境。

2.這次專案決定的的題材 Roguelite 加上第一人稱射擊，是十分好的決定，不但符合目前市場需求，也成功避開同類型遊戲的特點，雖然在開發上由於技術力的不足沒能製作完美，但也從中看到市場的可能性。

3.在開發中期由於沒能將團隊整合起來，導致開發進度大幅落後，也因此捨棄許多設計，雖在開發後期團隊穩定時將部分系統加回，但仍然與原企劃少一些系統，下次在參與專案時因更加注意團隊狀況，避免再次發生因團隊溝通或技術問題，導致開發進度落後。

第二節 展望

未來的展望：
透過這次的開發經驗，更加了解專案在執行時的種種問題，不論是硬體還是軟體，甚至是團隊內的問題，另外經過這次開發，團隊對於第一人稱射擊類型的遊戲有了更多的了解，像是槍枝後座力、腰射與開鏡的偏移、換彈動畫控制等等，以上資料必能對未來再次開發相似專案時給予極大幫助，最後希望能在上架平台獲得不錯的評價，以證明本次專案的價值。

參考文獻

Chris Lin,Website,Animation Rigging
(https://medium.com/@chrislin1015/unity%E6%89%8B%E6%9C%AD-animation-rigging-fa258201afee)

RaAlGhul,CSDN,FPS 遊戲開發之後坐力分析
(https://blog.csdn.net/RaAlGhul/article/details/80404586?utm_source=blogxgwz4)

ybhjx,CSDN,第一人稱射擊遊戲
(https://blog.csdn.net/ybhjx/article/details/92062917)