**主題: 遊戲集合**

**XXX XXX**

1. **設計理念**

在這個專題當中，我們希望自己能完成一個更加複雜的專案，並激發自己的創意，因此我(XXX)選擇在已經很經典的遊戲踩地雷上，加入了像是多人連線、擋死，競速等機制，讓它變得更有趣一些，也讓我熟悉稍微有些規模的專案應該要注意哪些地方。

1. **使用的函式庫**
2. **Tkinter:** 主要的視窗介面函式庫，提供了基本的 GUI 元件，透過這些元件的組合與排版，建立各遊戲所需的互動介面與視覺呈現。此外，它也提供事件綁定與主事件迴圈，讓按鍵、滑鼠點擊等互動能夠即時回饋，是遊戲與使用者操作之間的橋樑。
3. **Time:** 為了實現某些需要時間間隔控制的遊戲元素，例如延遲效果、計時、倒數，我們使用 time 函式庫來處理有關時間的操作。。
4. **Random:** random 在網路通訊當中，透過設定隨機種子，確保伺服器端與客戶端的遊戲版面一致，達到遊戲畫面同步的目的。
5. **Socket:** 網路通訊的核心，透過 TCP 連線建立伺服器和客戶端之間的雙向資料傳輸，使用它實作多人連線的機制，處理遊戲訊息的發送和接收。
6. **Json:** 為了讓複雜的資料能夠在網路中傳遞，我們將伺服器和客戶端之間傳送的資料轉換為 json 的格式，提高可讀性，也讓連線的兩端能明確地理解彼此的資料內容
7. **Threading:** 因為 GUI 和網路處理都需要立即回應，我們使用 threading 來實作非同步執行緒，像是接收訊息，傳送指令等網路任務，都會在背景執行，避免阻塞主函數，讓介面顯示保持流暢。
8. **Queue:** 在網路的訊息處理機制上使用 queue 紀錄需要處理的訊息，讓執行緒傳送的資料不會因為過於頻繁的傳送導致處理不來的問題。
9. **SQLite:** 作為後端的主要資料庫，負責儲存歷史紀錄等資訊，透過網路連線讓伺服器即時讀寫資料，實現多人同步與持久化儲存。
10. **困難與解決方法**

**遊戲一 :《 MineSweeper(單人與多人) 》**

我(XXX)負責開發踩地雷這款遊戲，最初的開發過程沒有遇到什麼問題，順利的做出了第一版，這個版本的踩地雷有最基礎的功能，左鍵翻開格子，右鍵放旗，假如翻開的地板是空的，沒有數字，就利用遞迴向外展開，開出更多格子。

在收到了一些反饋之後， 我加入了更多功能，最大的更新便是「快速開格」，在按下數字的時候，假如周圍一圈(也就是以自身為中心的3x3區塊)，有對應數字的旗幟，就翻開剩餘格子，我希望在按下時，能預先看到會翻開那些格子，為此，我將按下按鈕的動作分為兩個階段，按下與放開，按下時，能看見即將被翻開的格子塌陷，在放開時展開這些格子。另外，在打開遊戲的第一個區塊時，會有立刻踩到地雷的可能，或者打開數字，前者自然很破壞遊戲體驗，後者則是沒有提供玩家任何能立即使用的資訊，於是我稍微修改了擺放地雷的邏輯，將首次翻開的格子以及其周圍標示為不能放地雷的區域，在此之後，才擺放地雷，避免了上述問題。隨後我加入了三個調整難度的按鈕，讓遊戲能在按下按鈕之後產生不同大小的版面。以及重置遊戲的按鈕，到此，單人模式正式完成。

在開發到中途時，我便決定加入多人連線的功能，由於沒接觸過網路，相關的知識也不夠完備，在這一步我花了遠勝單人模式的時間去完成。

我開發了三次多人連線的版本，前兩次的失敗，在於單人版寫得不夠好，體現在它的擴充性差，以及耦合度高的問題，讓我在擴充時常遇到問題，忘記改變某個變數的數值，連線成功與失敗各自要做的事情，遊戲邏輯和 UI的更新交織在一起，稍有不慎，便可能導致遊戲出問題。

開發效率高不起來，因此我將程式碼全部重構，原本只有一個檔案的單人模式被我拆成了四個檔案，分別對應到「遊戲板上的格子」、「難度」、「遊戲板」「UI」，各司其職，大大減少了耦合度，並在此基礎上，又一次開始多人遊戲的開發。

有了前兩次失敗的經歷，這次的開發比較順利，第一步，我先做好了基本的網路通訊，讓兩台電腦在同一區域網下能夠互相連線，第二步，我在一台電腦做出操作之後，傳送一條文字訊息到對方身上，第三步，同步遊戲動作。一直到第三步之前都很順利，而在第三步的時候，我遇到了許多的問題。

首先是訊息的傳送，我應該要怎麼傳給對方，對方又要怎麼處理訊息? 最初我想要將整個板子當中每個格子的狀態傳送過去，並在另一個板子顯示出來，但這樣需要花很多時間處理資訊，延遲太高，既然兩邊都已經有了遊戲操作的基本邏輯，那為什麼不傳操作、以及對應的位置就好? 所以最後我使用 enum 定義了幾個操作，用json傳送操作，以及行列資料，成功同步了兩邊的操作。

這樣的想法很快就遇到了幾個問題，第一個問題是，兩邊板子的操作位置一樣，但產出的板面不同，於是我在傳送的資料中加入了隨機種子。

第二個問題，當操作太過頻繁，訊息來不及處理，會導致兩邊遊戲板狀態不同，雖說已經使用我用多執行緒的方式發送與接收訊息，但依舊會有處理不過來的問題，最後我使用 queue 去記下要發送以及接收的訊息，客戶端則會每經過一段時間，就主動將累積的遊戲訊息處理完畢，也就避免了忽略操作的問題。

第三個問題，就算有了 queue 和 thread去處理訊息，傳送的太過頻繁會封包沾黏的問題，也就是TCP在通訊時，假如長度不夠，便會將多條訊息合併傳送，此時就沒有辦法正確的讀取傳來的資料，解決這個問題有兩個方式，在每個訊息的前面寫上大小，或者加入固定的分隔符號，我選擇了比較好實現的分隔符號。

由於我把遊戲分成了好幾個類別，彼此有著父子階級的關係，這就衍生出了類別之間溝通的問題，舉例來說，NetworkManager類別才知道客戶端是否有成功連上，但MineSweeper主類別不知道，Networkmanager應該要怎麼通知MineSweeper客戶端的狀態，好讓MineSweeper能夠改變UI畫面，更新按鈕的狀態? 是要用一個變數紀錄? 還是說設定一個回調函數，並在初始化的時候指定? 抑或是存在更好的方法? 最初我使用回調函數，在NetworkManager發生某個事件，像是客戶端連上了，就觸發回調函數，回調函數就能去更新按鈕狀態。這個方法固然很好，可如果NetworkManager事件發生之後，需要通知不只一個類別，很明顯，設定多個回調函數不是一個好辦法，為了未來的擴展性。我將回調函數封裝了一層，讓一個事件可以觸發多個回調函數。

除了上述的技術問題，在遊戲玩法方面，我也曾有許多的想法，多人模式下，除了單純的競速之外，能不能變得更有趣些? 過久沒點開格子需不需要幫助自動翻開，點開地雷要不要改成增加時間懲罰? 而非直接失敗? 但考慮一段時間之後就打掉了這些想法。

最後我對競速稍微做了修改，從原本共用的計時器改為各自的計時器，第一下點開格子之後才開始計時，也就有了兩種玩法，等對方結束，或者在差不多的時間一同開始，有了些許心理戰的部分，要說美中不足的地方，就是只能在區域網下連線(但能透過Radmin VPN/Hamachi 等外部工具解決)，而且網路連線也會有延遲的存在。

自此，遊戲已經有了完整的流程，能夠無限的往下繼續進行，不過在幫隊友測試遊戲時，我看到他為遊戲加入了排行榜、以及按鍵盤迅速重置的功能，我因而受到啟發，為踩地雷也加入了這些機制，新增了計時器，讓單人遊戲變得更有趣一些。另外，在多人模式時常會有「二選一」的局面，也就是無法根據當前資訊推測出某些格子當中是否有地雷，容易讓遊戲陷入僵局，為此我加入了「擋死」的功能，用一個燈泡圖樣表示，當玩家第一次踩到地雷的時候，不會立即死亡，而是能繼續進行遊戲，避免了僵持狀態發生的機率。我將擋死的功能加入進兩個模式，讓這版踩地雷走出和經典版本不同的道路。

**遊戲二 :《 Zombie(打殭屍) 》**

我(XXX)主要負責開發Zombie遊戲的部分。在開發的過程中，我遇上了許多困難，包括排程問題、元件重疊問題、無法顯示背景、效能問題(會吃太多效能要簡化)等等，其中又以排程問題最為繁雜，大部分的時間都是在處理相關問題，也考驗著邏輯與信心。

一張含有 卡通, 圖解, 圖畫, 油畫 的圖片

AI 產生的內容可能不正確。圖一、網格邏輯

**1.殭屍圖像排列** 在這款遊戲的畫面設計上，我希望以簡潔直觀的方式來呈現玩家要擊打的殭屍，因此我設計了一個固定大小的顯示區域，也就是如圖一畫面中的「殭屍顯示格子」。我將畫面切割為五列三欄的網格，可以有效呈現殭屍的位置，同時不讓畫面過於擁擠。

這樣的格子規劃讓我可以清楚安排殭屍的出現位置，每列最多出現一隻殭屍，每次射擊的目標都會固定在最底下那列（Row 4），有助於玩家快速辨識與反應。為了在程式中對應這樣的排版，我使用了一個二維陣列專門用來儲存每一格的殭屍圖像。當某個位置沒有殭屍時，該格就設為 None，否則就放入對應的圖片物件。

**2.殭屍擊殺邏輯與更新**  
 當玩家按下方向鍵時，程式會比對目前最下方那一列殭屍的位置與玩家按的欄位是否一致。如果錯誤，則不進行更新，並顯示角色的失望語錄；若擊殺成功時，會將陣列的內容向下平移一格，並透過隨機生成在最上列補入新的殭屍。

此外，為了提升遊戲的可玩性，我加入了連擊(combo)與分數計算。每次擊中時，連擊會累加，分數則使用以下公式計算：

**self.score += int(1.1 \*\* self.combo)**

這個公式的設計目的是讓分數呈現遞增的回饋，一開始每次加分不多，但隨著combo值提高，得分成長會越來越快。例如combo為 1 時只加 1 分，combo為 10 時已可加約 2.5 分，combo為 20 時則超過 7 分，若失誤會使combo歸零，並觸發角色語音與短暫冷卻，防止亂按。這種「連擊越久越有價值」的設計，不僅要求玩家的速度，也要求打擊的精準度，加強挑戰性與成就感

**3.優化重複流程**

為了提升玩家反覆遊玩時的效率，我針對「開始」、「重新開始」與「選擇時間」這段重啟遊戲的操作流程進行優化，讓玩家能夠僅透過鍵盤就完成整個流程。

在「開始」與「重新開始」的部分，因為這兩個按鈕實際上是 canvas 上用create\_rectangle與create\_text畫出的元件，並不是真正的Tkinter元件，因此無法直接使用.invoke()等方法觸發。為了解決這個問題，我透過 bind("<Return>")將 Enter 鍵與自製的mouse\_click\_handler()綁定，並在其中模擬一個虛擬滑鼠事件，精確點擊按鈕的中心座標來觸發行為。

相較之下，「選擇時間」使用的是標準Tkinter的Toplevel視窗與按鈕元件，因此我可以直接使用focus\_set()指定預設按鈕，再透過 <Return> 自動觸發焦點元件的點擊，讓使用者在不操作滑鼠的情況下完成選擇。  
 這項設計整合了鍵盤輸入與滑鼠模擬，讓從「重新開始 → 選擇時間 → 開始遊戲」的整體流程可以透過連按Enter鍵無縫進行，大幅提升了遊戲迴圈的流暢體驗。

這樣的設計對玩家來說非常重要。由於本遊戲節奏快速，重玩次數高，若每次都需要滑鼠操作才能開始新一局，會造成體驗的中斷，更會影響手感。透過這套鍵盤優化機制，玩家可以在結束遊戲後，只需連按幾次 Enter 鍵即可重新開始一局，完全不需移動手指離開鍵盤。這種方便的操作流程，能有效維持專注度與遊戲沉浸感，並顯著提升整體使用體驗。

**4.優化重複流程**

剛開始放上素材的時候，是用傳統的widget方式，像是Label配上PhotoImage一張一張貼上去。結果一貼上去就出事了，角色圖片會重疊、遮到彼此，畫面亂七八糟。我那時候就是一個一個土法煉鋼得去調位置，改x座標、改y座標，想說「不讓他們重疊就好」。

所以畫面上雖然能動，但看起來就是一格一格生硬地擺著，角色之間空一大塊、不太自然，畫面也很陽春。但至少沒破圖，角色也都還看得清楚，我就暫時接受這樣的做法。  
 一直到我想加背景圖的時候，問題才開始爆發。只要角色圖放在背景上面，白邊就會露出來、圖片被裁掉一塊，或是被別的widget壓住，整個畫面像拼貼一樣破破的。這時我才意識到，用 widget 疊圖根本不適合這種需要圖層控制的遊戲畫面，這成了我改用Canvas的最後一根稻草。

改用Canvas之後，我把所有元件（角色圖、分數、倒數數字、combo效果）都用create\_image()放上去，變成一層一層自己控的圖。這樣我就可以自由調整誰在上面誰在下面，畫面也不會被widget限制住，只要素材是去背的，那就不會再出現白邊矩形背景(如圖二)的問題了。

另外還有一個問題是，有些效果是後來才出現的，像combo圖片或倒數數字。它們如果不特別處理，預設會被先出現的角色圖擋住，導致完全看不到。我解法是用canvas.tag\_raise()動態的控制圖層，確保每個圖案正確的上下關係。

整體來說，這其中經過了許多思考，因為原本是用傳統widget的方式實作，幾乎已經做了大部分的功能出來，如果要改的話，時間成本巨大，但受限於圖層(特別是背景不能好好顯示)的顯示，我下定決心重頭來過，將原本的功能全部改成使用canvas來寫，我也很慶幸當時有做這個決定，不僅圖層管理簡單，也沒有醜醜的矩形白背景，更讓畫面變得非常漂亮生動(圖三)。

**5.視窗與元件位置動態調整**

在實作階段，我遇到一個非常實際的困難：只要縮放視窗，或把遊戲交給別人用不同電腦執行，畫面上的元件就會亂跑，整個版面看起來非常混亂。最常見的情況是按鈕擠在奇怪的角落、殭屍疊在一起，甚至有些畫面會直接被截掉一半，連遊戲都沒辦法正常進行。

會出現這種狀況，是因為我一開始是用絕對座標去擺放所有東西。這種做法在開發用的解析度下沒問題，但不同電腦螢幕大小不一樣，有些人是 1080p，有些人是 4K，一旦進到別的解析度，整個版面配置就會大走鐘。

為了改善這個問題，我決定全面改用「比例座標」的方式來排版，意思是用 canvas\_width 和 canvas\_height 動態去計算每個元件應該擺在哪裡。舉個例子，像殭屍的位置我會這樣算：

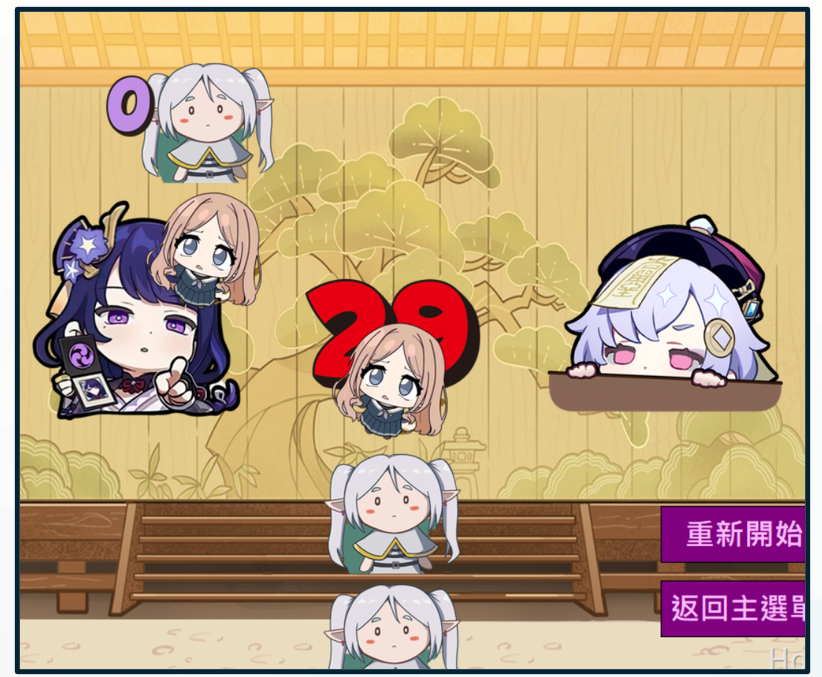
**start\_x = (canvas\_width - 殭屍總寬度) // 2**

這樣就能保證殭屍起始點永遠在畫面中間，其他殭屍再用固定間距向右擺放，整體排版就不會亂掉。不管視窗大小怎麼改，角色的位置都會自動調整到合理的位置。

一張含有 文字, 卡通, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

圖二、傳統做法，殭屍背景遮到角色



圖三、使用canvas，圖層美觀穩定

此外，為了應付使用者在遊戲進行中拖拉視窗大小的行為，我讓程式在每次 resize 時都會重新計算並重建整個畫面，包括角色圖、按鈕、特效等等，確保每個元件都維持在正確的比例位置上。

總結來說，這套「比例定位 + 動態重算」的機制，解決了不同螢幕解析度下的排版問題，也讓整體 UI 更加穩定，不會出現奇怪的畫面錯位，使用者體驗自然也提升了不少。

**6.分數與數字顯示：從傳統文字到素材圖的轉換**  
 一開始我只是很單純地用字串和 Label 來顯示分數或倒數計時，這種方式雖然寫起來快，但整體畫面感覺很陽春，跟我整體想要可愛、有風格的遊戲畫面完全不搭。

後來我就想：那我是不是可以用圖片來當數字顯示？這樣不只能套用主題風格，整體也會一致很多。那時我直覺想到：「如果我要做倒數，從30秒數到0，那我是不是得準備 0 到 30 的所有數字素材？」

這時我同學直接提醒我：「你幹嘛要準備 0 到 30？你只要準備 0 到 9 的圖片就夠了啊，用字串拆開來一位一位去對應不就好了？」

我那時候突然有種開竅的感覺，才發現這其實就是把數字轉成字串，然後每一位對應一張圖片貼上去就好。這樣不管是分數、倒數、combo效果，只要準備 0～9 的素材就能全部搞定，還能隨時拼出任意位數的顯示。

後來還遇到分數太長的問題，會導致畫面撐破，我再額外做了單位縮寫（例如 1649b 代表一千六百四十九萬），讓整體版面不會被數字撐爆。這整套做下來，比原本的Label顯示靈活很多，而且畫面風格也大幅提升。

**7.動畫**

為了讓遊戲畫面更有打擊感，我特別幫殭屍加上兩種動畫：一種是miss時的彈跳動畫，另一種是擊殺後的整體殭屍的下滑動畫。  
**(1)殭屍跳**  
 一開始我只是讓殭屍 miss 的時候出現紅色背景，但這樣看起來太呆板了，沒有真實感。所以我改成讓殭屍「跳一下再落地」，看起來像被嚇到一樣彈了一下。這部分我用了簡單的拋物線公式去模擬彈跳軌跡。做法是寫一個函數，動畫設為一秒，用來算出 Y 軸位移。實際公式是：

**def easing(t):  
 return -4 \* height \* (t - 0.5) \*\* 2 + height**

這個公式本質上是一個開口向下的拋物線，其核心是 -4 \* height \* (t - 0.5)² + height，其中 t 是時間（殭屍跳會跳一秒），height 是最大跳躍高度。當 t = 0 或 t = 1 時，(t - 0.5)² = 0.25，整體輸出變成 -height + height = 0，也就是回到原本的 Y 座標（地面）。當 t = 0.5 時，(t - 0.5)² = 0，輸出為height，也就是跳到最高點。

由於畫面中的 Y 軸是向下為正（數值越大位置越低），這條拋物線會讓殭屍從原地往上跳（Y 值變小），然後再落回原地（Y 值變大），視覺上就會像是「蹦一下再落地」的動作，呈現出自然的彈跳曲線。  
**(2)殭屍下移** 當玩家擊殺殭屍後，我希望整排殭屍能夠一起往下滑一格，而不是直接瞬間跳位置，這樣畫面會比較順、節奏感也比較好。這段動畫我沒有用外部函式庫，而是直接自己透過 after() 來實作。

我寫了一個play\_zombie\_down\_animation()函式，它會每幾毫秒執行一次，逐幀更新殭屍的位置。整體動畫我分成move\_frames幀（我設的是 10 幀），每一幀都會把殭屍的 Y 座標往下加一小段距離（也就是grid\_height // move\_frames）。

每一次更新都是直接用canvas.coords()把每隻殭屍的座標重新設定，而不是用 move() 疊加，這樣可以避免畫面抖動或誤差累積。

整段動畫使殭屍平滑地下移，看起來更自然，也幫助玩家更清楚地接收到「這一操作已經結束了」的訊號。

**8.排程處理**

整個專案中我遇到最多的困難與問題就是排程。畫面中有太多不同的after排程，像是倒數、動畫、combo、分數更新、殭屍下移等等。只要漏掉一個沒取消，就會造成畫面殘影或動畫重疊。像是重新開始遊戲後，如果之前的倒數排程還在跑，數字就會一直堆上來。這種問題需要使用after\_cancel()手動取消排程，再用canvas.delete()清除畫面殘留，才能真正解決。

排程問題雖然不起眼，但花掉我最多debug時間，是整個專案中最複雜也最容易出錯的部分。

**【雲端排行榜】**

踩地雷跟打殭屍都是採用雲端排行榜，共用同一套tunnel來上傳資料，個別的資料庫也都是在同一個機器上運行。

**困難一:本地無法全球化**  
在做排行榜的時候，我一開始是把資料庫直接放在本地，這樣寫起來簡單，也可以正常存資料。但問題是：這個遊戲設計初衷是希望可以給全球玩家玩的，排行榜寫進本地後，別人根本看不到。就算有人破了紀錄，也只存在自己電腦裡，別人完全不知道發生了什麼。

**解決辦法:**建立一個共用的中心化雲端資料庫，不管從哪一個裝置啟動遊戲，讀取的排行榜都是同一個，也就是雲上的這一個。  
**困難二:資料庫更新問題**

一開始剛上雲時還沒發現，但過了一陣子回來查看時馬上出現一個很致命的 bug：只要伺服器一關、或重啟容器，排行榜的資料就會整個歸零，變成初始化的狀態。

我一開始完全不知道為什麼會這樣，還以為是程式哪裡沒寫好。後來才發現，我把 .db 檔案直接包在Docker image裡面，而Docker image只要重建，裡面的資料就會被還原，之前寫進去的資料根本沒有存起來。這問題當時害我一直以為排行榜壞了，debug超久。

**解決辦法:**把資料庫拉出 image，用 Docker volume 掛載。這樣資料會獨立存放在 container 外面，就算 image 重建也不會被覆蓋。排行榜寫進去的內容才終於不會消失，跨機器也能共用。

在這段過程中，我真的學到很多，還特別去查雲端容器的理論，學到原來雲端容器需要image-volume這樣一個架構，image負責存取唯讀資料，例如運行程式之類的，而volume需要存取會被更新的資料，像是這個專題的資料庫。



圖四、雲端架構

由於排行榜需要多人共用，所以我將資料庫移到雲端伺服器上，讓所有玩家都能讀寫同一份紀錄(如圖四)。

* 前端遊戲仍然是本地執行，運行反應比較快
* 排行榜則放在雲端伺服器，由 Messaging API 橋接，統一讀寫資料庫

整體架構分成左右兩邊：

左邊是本地裝置，運行我們的遊戲合集GameMenu，GameMenu裡面有我們的不同遊戲。這些都是玩家電腦上直接執行的程式，互動即時不卡頓。

右邊是雲端部分，我把 Flask 程式（app.py）和兩個 SQLite 資料庫（Zombie.db、MineSweeper.db）包進一個 Docker container，然後部署在 Fly.io 上。只要玩家有連網，就能透過 Messaging API 傳送請求給伺服器，伺服器再統一幫忙處理資料庫的讀寫。該機器可以隨時維持運行，一段時間沒有request就會休眠以節省資源，但不會影響到資料庫，因為我的資料庫是存在volume裡，再次開啟時就會從image取得運行程式並執行，然後在從volume裡取得正確、已被更新的資料庫。

1. **遊戲玩法說明**

**MineSweeper踩地雷(單人版)**

在遊戲畫面中，玩家會看到如圖所示的一塊遊戲板，以及下方設定難度的按鈕，遊戲板塊由圖一上的方塊組合而成。

圖六、MineSweeper遊戲畫面



圖五、遊戲版會出現的方塊

代表尚未翻開的格子，當所有為翻開的格子被翻開時，遊戲通關。

代表翻開的格子，且以這格為中心的 3x3 區塊當中，都沒有地雷存在。

同樣代表翻開的格子，中間寫的數字為 1 ~ 8，代表以該格為中心點的 3x3 區塊，都沒有地雷的存在。

是玩家主動打上的標記，只能標在未翻開的格子上，代表玩家認為這個格子有存在地雷，不應該翻開，作為後續遊戲的提醒。

代表未爆炸的地雷，在踩到地雷時，會將所有格子翻開，告知玩家地雷實際的位置所在。

代表爆炸的地雷，當玩家翻開含有地雷的格子，且燈泡不亮，地雷爆炸，遊戲結束，翻開所有的格子，並告知玩家踩到地雷。

可以視為額外生命，當玩家翻開地雷時，假如燈泡有亮起，地雷不會爆炸，燈泡熄滅，玩家可主動關閉燈泡。

玩家要完成的目的，就是翻開所有非地雷的格子，並在過程中利用翻開格子的資訊，推論地雷在哪裡，贏下遊戲。

在圖二當中，除了遊戲版之外，下方還有簡單、普通、困難、重置遊戲四個按鈕，難度的詳細設定如下

1. **簡單:** 遊戲版面 9 x 9個格子，包含 10 個地雷
2. **普通:** 遊戲版面 16 x 16個格子，包含 40 個地雷
3. **困難:** 遊戲版面 30 x 16個格子，包含 99 個地雷

而重置遊戲會保留當前的難度選擇，將所有格子都變回未翻開的狀態，可以在遊戲進行到一半的時候按下。

**遊戲玩法**

1. **翻開格子:** 對未翻開的格子按下滑鼠左鍵，假如當前選取到翻開尚未翻開的格子，便會翻開該格，若此時翻開的是地雷，地雷爆炸，遊戲結束。

當遊戲翻開的第一個格子的時候，以該格為中心的 3x3 區塊，都不會有地雷出現，避免在遊戲沒有提供資訊的情況下就輸掉遊戲。

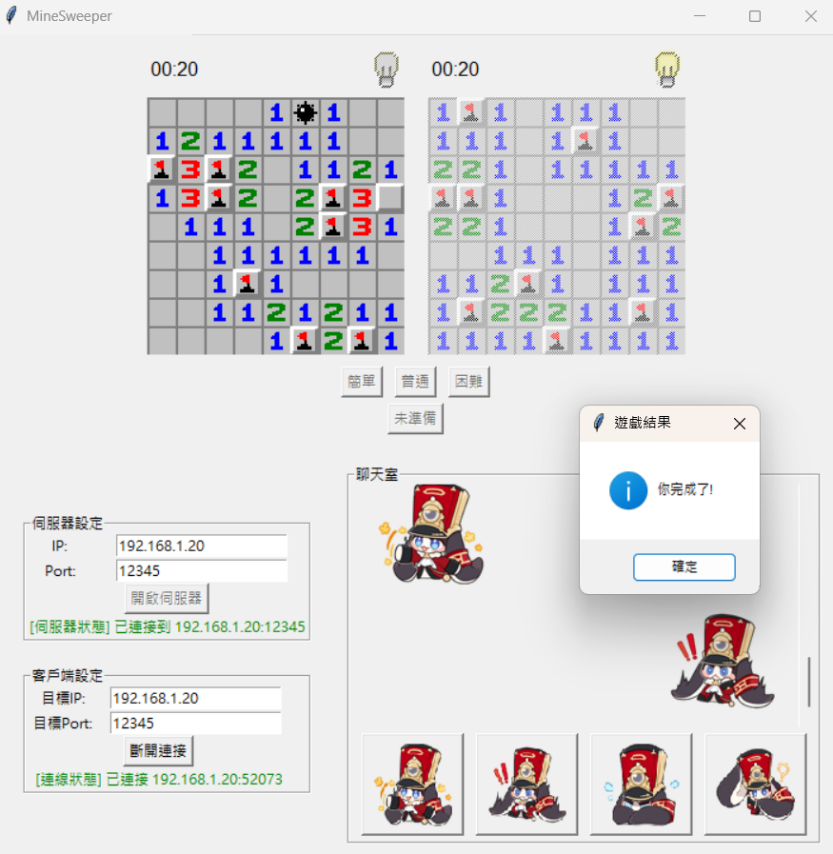
1. **標記旗子:** 對未翻開的格子按下滑鼠右鍵，假如當前選取道尚未翻開的格子，就會用旗子標記它，再次左鍵便能取消標記。

對被標記的格子按下左鍵，並不會翻開該格，從而避免玩家誤觸，也方便玩家在遊戲進行中利用標記加快自己遊玩的速度，

1. **數字展開:** 對含有數字的格子按下左鍵，假如以自己為中心的 3x3區域的旗子數量，有對上格子的數字，就自動翻開該區域內，除了有旗的格子外，所有尚未被翻開的格子。
2. **燈泡擋死:** 板塊右上角有一個燈泡按鈕，當遊戲開始的時候亮起，玩家可自行開關，當燈泡亮起的時候，假如玩家踩到地雷，燈泡熄滅，此時遊戲不會結束，而是繼續進行下去。會有這個機制存在的原因，是為了讓玩家在單人模式下有跟多人模式相同的體驗。之所以在多人加入這個機制，是為了避免陷入僵局(推測不出正確的地雷位置，只能靠猜的時候)。

**MineSweeper踩地雷(多人版)**

多人版本加入了連線的功能，兩台電腦必須要位於同一個區域網下，伺服器會先填入電腦的 ip 位置，預設的port為12345，可以隨意修改，在設定下方會顯示當前的連線狀態。



圖七、踩地雷多人版介面

客戶端輸入 ip 連上電腦之後，難度與準備按鈕便可按下，當兩方都準備好之後，倒數三秒之後開始遊戲。雙方遊戲版面上方的計時器，會在各自第一次點擊之後開始計算，一直到解開整個遊戲版面之後結束。當兩方遊戲都結束之後，聊天室會出現最終各自花費的時間，重置遊戲版面，回到遊戲最初的狀態。遊戲中途，遊戲雙方可以透過表情符號交流。

**Zombie(打殭屍)**

這是一款反應型射擊遊戲，玩家需要使用方向鍵擊殺從畫面上方出現的殭屍(如圖八)。  
遊戲開始後，殭屍會從畫面上方一格一格往下移動。每一隻殭屍的出現位置都對應鍵盤上的方向鍵：  
 ←：左邊的殭屍  
 ↓：中間的殭屍  
 →：右邊的殭屍  
玩家要在殭屍抵達畫面下方前，按下對應方向鍵來擊殺殭屍。只要按對了，就會得分並出現擊殺動畫。如果按錯，則會觸發 miss 效果，combo 歸零並顯示角色語音懲罰。  
隨著遊戲進行，殭屍會持續出現，速度也會越來越快，考驗玩家的反應與手速。遊戲右下方有「重新開始」按鈕，隨時可以重玩。  
最終的得分會記錄到排行榜中，並與其他玩家比拼最高紀錄

  
圖八、Zombie以方向鍵進行擊殺

1. **結論**

在這次的專題中，我們結合了程式邏輯、網路通訊、圖像處理與使用者體驗等多方面的能力，成功打造出一款整合型的遊戲集合平台，並分別完成了《MineSweeper 踩地雷（單人與多人）》與《Zombie 打殭屍》這兩款特色鮮明的遊戲作品，加上網路多人連線設計，與資料庫雲端實現，我們不僅讓遊戲突破了單機限制，也真正實踐了「分散式架構」的概念。透過 Socket 通訊與 Flask 伺服器的整合，玩家能夠在區域網內連線對戰，並同步彼此的操作狀態；而雲端排行榜的建立，則讓所有裝置能共用同一份資料庫，即使在不同時間與地點，也能持續追蹤個人成績與競爭排名。這些功能的實作，讓我們更深入了解前後端協作的實務流程，也為我們未來開發跨平台或大型專案打下了重要的基礎。)。

整體開發過程雖然充滿挑戰，但也讓我們從中學到了許多課本之外、實作層面的重要技能與觀念。

1. **心得 by S1154041 XXX**

這次的專題當中，我們做出了一個遊戲的合集，雖說兩個遊戲都是復刻現有的遊戲，但至少有做出一些變化，不至於完全相同。

踩地雷涉及到了網路的知識，隊友做的遊戲涉及到了雲端容器，也因為跨足到了不熟悉的領域，在這些部分遇到了問題是最多的。成品能讓玩家透過網路連線對戰，連上雲端，實時更新排行榜，和前後端的觀念很相似，或許這份經驗在未來能作為開發更大型專案的助力。

另外，開發過程自然會和其他組別的人去討論意見，也會看到其他人做了什麼樣的主題。心理會升起一些比較心態，想「做些很厲害的東西，讓同儕刮目相看」， 是在製作過程中隱藏的動力，也正是因為這樣，沒修過網路相關課程的我，硬是把連線功能做出來。

除了其他組別激起好勝心之外，隊友XXX也在過程中跟我討論了很多開發方向的方面，找到了作為一種玩法的Bug，要不要修? 遊戲還可以怎麼改?一直到鄰近截止日期，還有許多的想法沒有實現，像是踩地雷在單人模式下，加入每個排行榜上其他玩家的遊戲過程。又或是多人模式加入更多種技能，增強「對戰」的感覺。

要說對這個專題不滿意的部分，我想就是主題的選擇，「踩地雷」終究不是一個會讓人耳目一新的主題，可要自己從頭想一個遊戲又太困難，但除此之外，這依舊是個能讓我滿意的作品。一言以蔽之，專題學到很多東西，不論是在知識上，還是在溝通上。