# I. Introduction

本程式為以 C++所構成之單人文字冒險遊戲，運用物件打包、物件繼承、Virtual function 等特性進行實作；使用者在遊戲中扮演地下城冒險者，透過鍵盤輸入來控制遊戲進行，本程式除了**達成所有要求之基本功能**，為進一步增加遊戲趣味性再加入了額外功能，報告中將詳細介紹。

# II. Implementation & Discussion

## 1. 程式執行概要

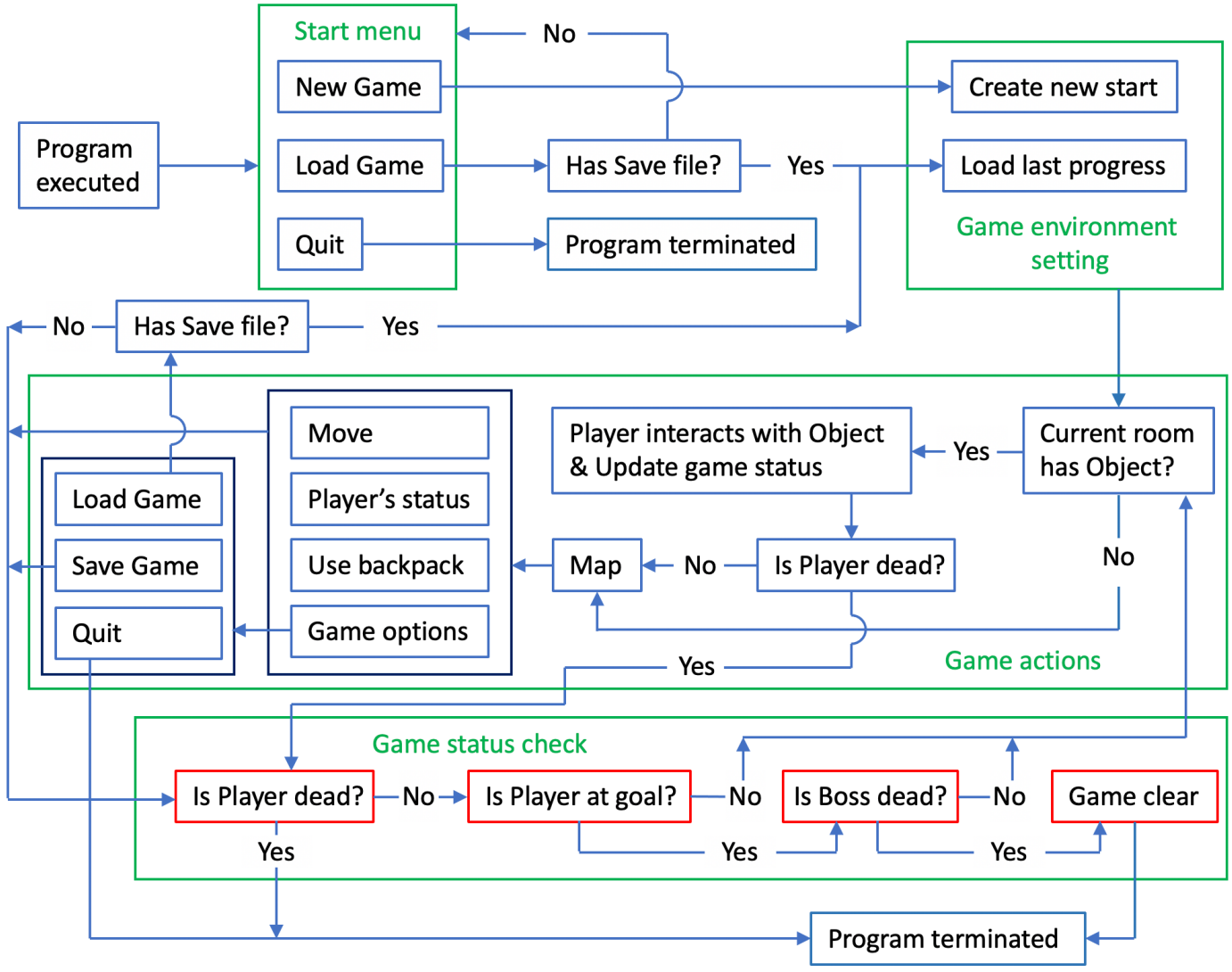


圖 1: 程式運作流程圖，箭頭指向代表程式進行方向

圖 1 為程式運作流程圖，主要包含 4 項功能：**開始選單(Start menu)**、**遊戲環境參數設定(Game environment setting)**、**遊戲動作(Game actions)**、**遊戲狀態檢查(Game status check)**，下面分別對以上功能介紹：

### (a) 開始選單 & 遊戲環境參數設定：

開始選單為程式執行後讓使用者選擇哪種遊戲環境參數設定；選擇新遊戲(New Game)的情況下，會指示程式以預設數值來設定，大多數為固定數值，一些物件參數使用了隨機函數來生成；選擇讀檔

(Load Game)的情況，會先判別是否有存檔資料(Has save file?)，有則用存檔數值來設定遊戲環境，沒有則讓使用者重新在開始選單做選擇；選擇離開(Quit)則會讓程式直接終止。

### (b) 遊戲動作：

此功能主導了大部分遊戲的進行；首先會檢查角色(Player)目前房間是否含有物件(Object)，有物件的情況下強制使角色與物件互動；為了單純處理，只讓每個房間最多含一個物件，物件包含三種：怪物(Monster)、寶箱(Chest)、商店(Shop)，在對不同物件與不同互動場景下，角色與物件的數值狀態均進行更新；互動後最重要的是檢查角色是否死亡(Is Player dead?)，是的話無需進行後續動作，直接轉移至遊戲狀態檢查。

在沒有物件或角色為死亡的情況下，會先顯示地圖(Map)，讓使用者在後續選擇移動方向較為便利，再來讓使用者選擇移動(Move)、顯示角色狀態(Player’ s status)、使用背包(Use backpack)、遊戲選單(Game options)，選擇選項後會完成對應動作，再轉移至遊戲狀態檢查；遊戲選單中含有存檔(Save game)、讀檔(Load game)、離開(Quit)三個選項；存檔無論成功與否，均會轉移至遊戲狀態檢查；讀檔成功會轉至遊戲環境參數設定，失敗則轉至遊戲狀態檢查；離開是直接終止程式。

### (c) 遊戲狀態檢查：

此功能用來判斷與控制遊戲是否結束；遊戲結束分成通關狀態與非通關狀態，前者條件為角色擊敗最終魔王(Boss)且抵達終點房間，後者為角色死亡；功能判斷遊戲結束則會終止程式，遊戲未結束則會轉移至遊戲動作功能。

## 2. 程式實作細節

圖 2 呈現了本程式所有物件以及對應之 Class Diagram；一個地下城冒險應含有玩家、怪物、寶箱、商店、道具與迷宮等基本要素，這裡玩家以 class Player 實作、怪物以 class Monster 實作、商店以 class NPC 實作，寶箱與道具以 class Item 實作，迷宮以 class Room 實作，以上物件最後都會包含在 class Dungeon 中生成，而控制遊戲存讀檔以 class Record 實作，同樣也會包含在 class Dungeon 中生成；最後，只要 main function 生成 class Dungeon，再呼叫執行地下城遊戲的 function，即可執行遊戲。下面會分別說明：**(a) 各 class 實作概要**，**(b) 基本功能實作細節**，**(c) 額外功能實作細節**。

### (a) 各 class 實作概要：

**a-1. Object, Item, GameCharacter：**

Object 為所有遊戲角色與道具的原型，含有共通屬性參數(attribute parameter)：name 和 tag，後續繼承 Object 的 class 可以再另外設定 name，例如玩家的名字；tag 雖然在遊戲進行中不會顯示，但可在存檔上作為辨識資訊；Object 的操作方法(operation method): triggerEvent 設定為 pure virtual function，能讓繼承 Object 的 class 各自去 override 並定義自己要從事的行為，程式執行中可透過 late binding 自動呼叫對應 class 的 triggerEvent，在此，Object 被設計為 abstract class，不可被實體化；後續繼承 Object 的 class A 想與繼承 Object 的 class B 互動，則可使用指令：A.triggerEvent(pointer B)，triggerEvent 的返回值為 bool，可以用來代表互動成功與否或其他。

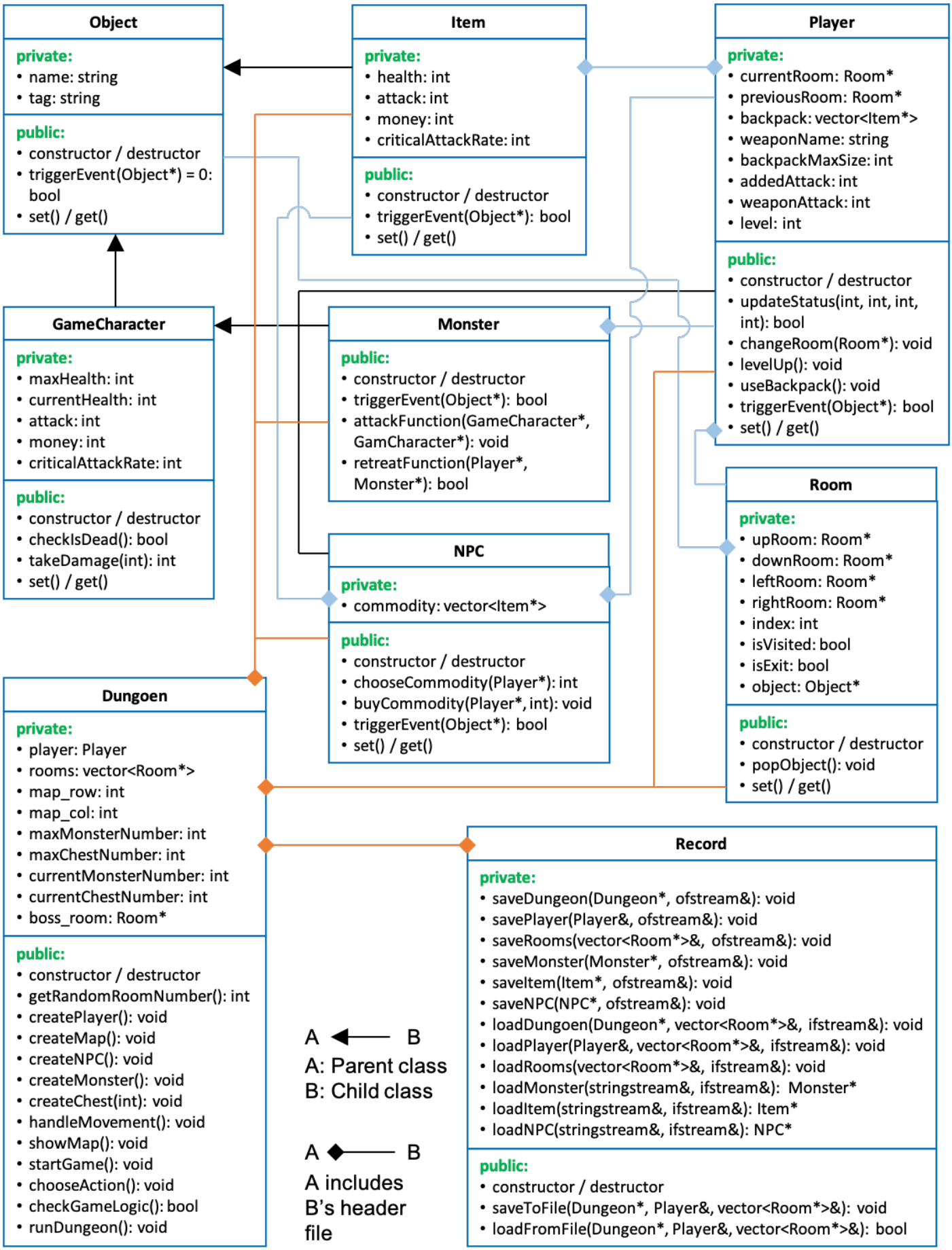


圖 2: 程式中各物件之 Class Diagram

Item 繼承了 Object 用來生成各種道具，道具設計上是讓玩家角色的能力值提升，因此操作方法 triggerEvent 內定義了玩家使用道具後，如何提升玩家角色的能力值，包含血量、攻擊力、錢、爆擊率，以上四項也就作為了 Item 的屬性參數(health, attack, money, criticalAttackRate)，triggerEvent 的傳入參數為 Player pointer；本遊戲中生成的 Item 分成三種：寶箱道具、回復藥水、武器，triggerEvent 中透過辨識道具的 Object::name 來分別處理與玩家的互動。

GameCharacter 繼承了 Object，作為所有遊戲角色的原型，本遊戲設計了三種遊戲角色：玩家、怪物、商店；商店主要為販賣道具給玩家，不會攻擊也不會死亡消失；玩家和怪物可以互相攻擊對方，怪物死亡會消失，玩家死亡則會導向遊戲結束，因此 GameCharacter 的屬性參數和操作方法主要為玩家和怪物的共同屬性參數和操作方法；屬性參數包含：最大血量(maxHealth)、目前血量(currentHealth)、攻擊力(attack)、錢(money)、爆擊率(criticalAttackRate)；操作方法包含兩項：1. 決定角色是否死亡(checkIsDead)，此項檢查角色目前血量是否為 0 來決定角色是否死亡，2. 承受攻擊(takeDamage)，將對手的攻擊值作為傳入參數，減少角色目前血量；同樣 GameCharacter 被設計為 abstract class，不用 override triggerEvent，也不可被實體化。

**a-2. Room：**

Room 沒有繼承其他 class；地下城的空間是由一個個 Room 實體相互串接所構成，因此屬性參數中用 4 個 Room pointer 來指向被串接的 Room，分別代表在上、下、左、右、四個方位的房間，對應屬性參數分別為 upRoom, downRoom, leftRoom, rightRoom；一個房間設計最多含有一個道具或遊戲角色，因此以一個 Object pointer(屬性參數 object)指向道具或遊戲角色實體；玩家與道具或遊戲角色實體互動完後，若要消滅該實體，可執行操作方法 popObject；由於每此執行程式所生成之 Room 實體記憶體位置並非固定，無法作為存讀檔的資訊，此問題能透過讓每個 Room 擁有固定編號(屬性參數 index)解決；遊戲通關條件之一為玩家抵達終點 Room，因此透過屬性參數 isExit 決定此 Room 是否為終點，而屬性參數 isVisited 決定 Room 的資訊是否呈現在地圖上。

**a-3. Player：**

Player 繼承了 GameCharacter，此為使用者主要操控之遊戲角色；第一個功能是能在地下城空間移動，因此使用 2 個 Room pointer 參數 currentRoom 和 previousRoom，分別記錄角色目前與上一次移動的位置資訊，這兩個參數的互相搭配是為了讓戰鬥撤退功能順利執行，避免發生死迴圈；每次執行移動便會執行操作方法 changeRoom

第二個功能是讓 Player 能透過撿寶箱道具、使用回復藥水或裝備武器來提升 Player 能力數值，updateStatus 這個操作方法用來改變 Player 能力數值，傳入參數則放入使用道具的屬性參數，return 值為 bool，true 代表 Player 能力數值改變成功，false 則為失敗

第三個功能是查看 Player 能力數值，定義在操作方法 triggerEvent 中，基本上不用與其他物件互動，因此傳入參數為 null pointer，操作方法內基本上就是呈現 Object, GameCharacter, Player 的屬性參數。

第四個功能為使用背包道具，由 Player 屬性參數 backpack, backpackMaxSize 和操作方法 useBackpack 組合達成，backpack 以 vector 乘載 Item pointer，目的是讓放入、取出、移動背包中的道具較有效率，而 backpackMaxSize 可以控制背包內存有道具上限；在 useBackpack 中，會

先檢查背包內是否有道具，無則離開，有則列出所有道具讓使用者選擇，使用者可以選擇使用道具或丟棄道具。

第五個功能是控制 Player 等級提升，屬性參數 level 記錄 Player 目前等級，操作方法 levelUP 則控制等級提升時 Player 能力數值如何提升。

**a-4. Monster：**

Monster 繼承了 GameCharacter，作為地下城中冒險者的敵人；Monster 能力數值基本上都定義在 GameCharacter 的屬性參數，triggerEvent 定義了如何與 Player 對戰，傳入參數為 Player pointer；對戰中可能發生事件：1. 角色攻擊(Monster 對 Player 或 Player 對 Monster)、2. Player 查看自身狀態、3. Player 使用回血道具、4. Player 進行撤退；第一項事件定義在操作方法 attackFunction 中，傳入參數第一個 GameCharacter pointer 代表攻擊方，第二個 GameCharacter pointer 代表被攻擊方，攻擊方給予的攻擊值加入了額外功能隨機函數，有機會給對手造成 2 倍傷害；第二項事件和第三項事件分別使用 Player::triggerEvent 和 Player::useBackpack 即可；第四項事件定義在操作方法 retreatFunction 中，同樣加入了額外功能隨機函數決定撤退成功與否，失敗的情況下 Player 會受到一次 Monster 的攻擊。

**a-5. NPC：**

NPC 繼承了 GameCharacter，作為地下城中的道具商店並販賣兩種商品：武器與回復藥水，因此擁有屬性參數 commodity 來承裝道具，與 Player::backpack 相同，以 vector 乘載 Item pointer；triggerEvent 定義了如何販賣道具給 Player，傳入參數為 Player pointer；商店與 Player 的互動包含了:1. 陳列商品、2. 玩家選擇商品、3. 玩家購買商品，前兩項定義在操作方法 chooseCommodity，最後一項定義在操作方法 buyCommodity。

**a-6. Dungeon & Record：**

Record 沒有繼承其他 class，主要功能為存讀取遊戲資訊，因此由多個操作方法組成；需要存讀檔時，會分別執行 saveToFile 和 loadToFile；前者 function 內部會再執行 saveDungeon, savePlayer, saveRooms, saveMonster, saveItem, saveNPC，將必要遊戲資訊輸出至 txt 文件；在有存檔之 txt 文件情況下，後者 function 內部會執行 loadDungeon, loadPlayer, loadRooms, loadMonster, loadItem, loadNPC 讀取 txt 文件，更新既有物件之屬性參數或生成新物件。

Dungeon 也沒有繼承其他 class，前面介紹了 Dungeon 含有玩家與迷宮，迷宮中有寶箱、商店、怪物，因此用屬性參數 player 和 rooms 分別代表玩家和迷宮，前者為 class Player，後者以 vector 乘載 Room pointer；本遊戲設計的迷宮為數個房間排列成矩形，因此用屬性參數 map_row 和 map_col 記錄矩形邊長；前面提到，遊戲通關條件為抵達終點房間與擊敗最終魔王，這裡在生成地下城時，會使終點與魔王在不同房間，為使確認魔王生存較有效率，用了屬性參數 boss_room 來存放魔王所在房間的 Room pointer；魔王在地下城產生時便只會創造一個，消滅後便不會在生成，而地圖上的寶箱與魔王之外的怪物在消失後有機會再生成，透過屬性參數目前寶箱數量(currentChestNumber)、最大寶箱數量(maxChestNumber)、目前魔王之外怪物數量(currentMonsterNumber)、最大魔王之外怪物數量(maxMonsterNumber)互相搭配決定。

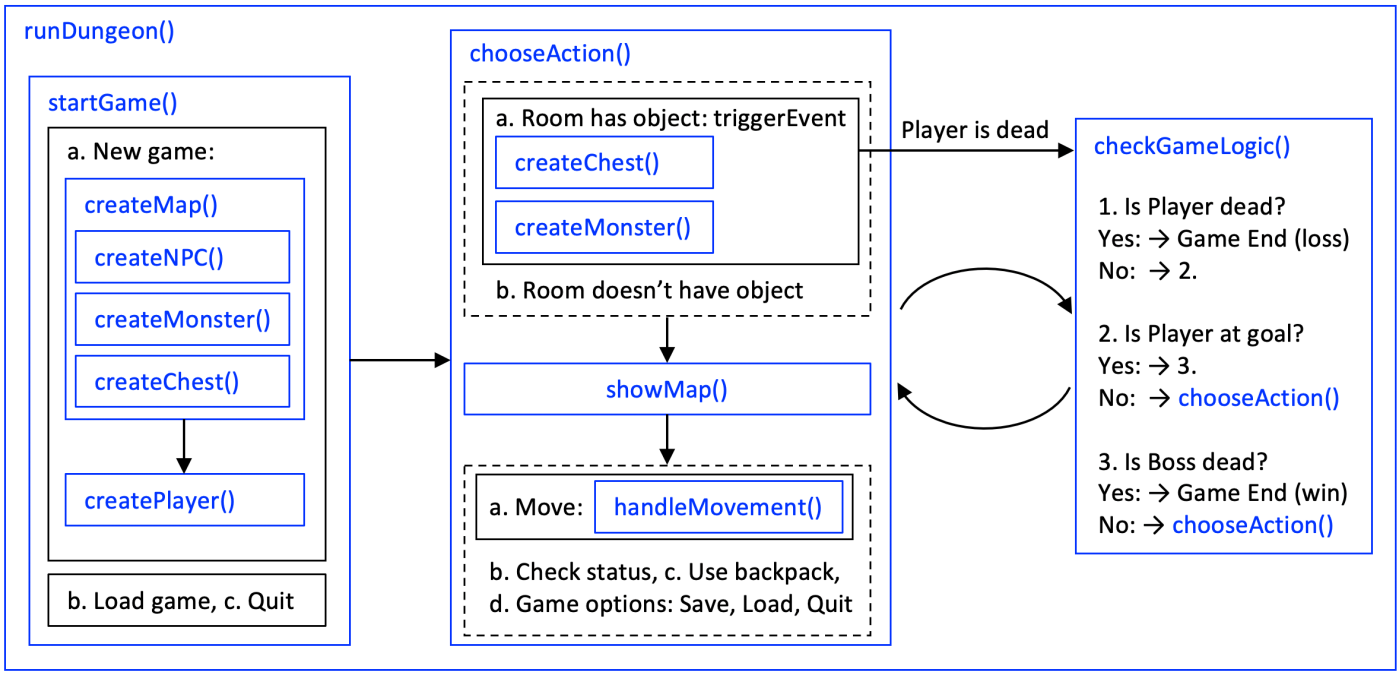


圖 3: 透過 class Dungeon 之操作方法所構成之遊戲運行示意圖

圖 3 呈現了如何透過 Dungeon 的操作方法構建整個遊戲，首先在 main function 生成 Dungeon 實體後，整個遊戲在 runDungeon 這個操作方法中進行；runDungeon 內會先執行 startGame 設定遊戲環境，再來進入 chooseAction 和 checkGameLogic 共同構成的無限迴圈；使用者若在 startGame 中選擇新遊戲(a. New game)，程式會先執行 createMap 生成初始迷宮，再執行 createPlayer 讓使用者設定玩家角色姓名，再將角色放置於迷宮；createMap 中所發生的事件依序為：1. 生成房間矩形陣列並彼此串接形成迷宮，2. 設定終點位置，3. 生成一個 Boss 放置於迷宮 4. createNPC 生成含有數個道具的一個商店放置於迷宮，5. createMonster 生成數個敵人放置於迷宮，6. createChest 生成數個道具放置於迷宮；chooseAction 主要控制遊戲的進行，對使用者的選擇做出反應；checkGameLogic 會在 chooseAction 執行完畢後接著執行，用來判斷目前狀態是否要回到 chooseAction 繼續進行遊戲；在 chooseAction 中，首先是玩家角色與目前房間之物件互動，即在有物件的情況下執行 triggerEvent，互動後首先檢查玩家角色是否死亡，若死亡則立即轉至 checkGameLogic，若否，在特定條件達成下會執行 createChest 和 createMonster 再生成寶箱和敵人放置於迷宮中；再來，會透過 showMap 呈現地圖讓玩家後續在移動選擇上更為便利；最後，會讓使用者進行角色操控選擇，其中會以 handleMovement 控制玩家角色的移動；在圖 3 中沒有呈現的操作方法為 getRandomRoomNumber，這個方法用來產生隨間房間 index，搭配在終點、Boss、商店、敵人、寶箱、玩家開始位置設定上。

### a-7. 各 class 之 Constructor & Destructor 與其他：

在 constructor 的編寫上，除了 class Dungeon 和 class Record，均使用 constructor with default argument，同時兼顧有傳入參數時的 constructor 和沒傳入參數時當作 default constructor 使用，需注意此寫法下不可再額外寫 default constructor，會造成 compiler error；使用 constructor with default argument 寫法的各 class，均設定了生成時各屬性參數的預設值；而 class Dungeon 和 class Record 沒有需要傳入不同參數設定屬性的情形，因此均使用 default constructor；在 class 屬性方法含有 vector 時(Dungeon::rooms, Player::backpack, NPC::commodity)，會使用

vector.reserve()直接設定該 class 的 vector 最大使用空間，避免後續進行 push_back()時 vector 重複進行 memory allocation 以及 vector 內 element 的複製與搬動；在 destructor 編寫上，有繼承 class Object 的 class 均須定義 virtual destructor，避免物件消滅時出現錯誤；若 class 中有使用 pointer 指向 dynamic memory allocation 生成之物件(包含 class Dungeon, class Room, class Player, class NPC)，在 destructor 中均執行了 release memory 以避免 memory leak 發生。

在 header file 有互相 include 的情況下，如 Player.h 和 Item.h、Dungeon.h 和 Record.h，需在自己的 header file 內加入對方 class 的 forward declaration，避免 compiler 產生 undeclared identifier 的 error。

## (b) 基本功能實作細節：

### b-1. Movement：

此功能讓玩家能主動在迷宮中移動，以 Dungeon::handleMovement()達成，在此操作方法中，一律會出現上下左右四個方向讓使用者選擇，然而，在使用者選擇對應方向後會檢查該方向是否存在房間，若有，則執行 Player::changeRoom(Room*)進行移動，若無，則顯示訊息讓使用者再次選擇；Player::changeRoom(Room*)的傳入參數為目的地房間的 pointer，function 內會先將 Player::currentRoom 存入 Player::previousRoom，再將傳入參數存入 Player::currentRoom，即完成玩家角色移動。

在 **a-6.**提到，房間若有物件會直接執行 triggerEvent 讓 Player 與物件互動，表 1 中整理了玩家遭遇不同物件、執行 triggerEvent 後不同 return 值所對應意義，其中有觸發強制移動，直接執行 Player::changeRoom(Room*)；例如，作業要求戰鬥撤退時玩家強制移動至前一步房間，本遊戲也設定了讓 Player 離開商店會強制移動至前一步房間，因此在 Dungeon::chooseAction()中一開始設計 Object::triggerEvent 的返回值為 false 則執行強制移動至上一步，但此設計時遇到了如圖 4 的「移動卡死」問題，例如，當兩個 O 均為 NPC 時就會造成 Player 無法離開地圖角落，或是 Player

| triggerEvent return value | class Item | class Monster | class NPC |
|---|---|---|---|
| true | Player 使用 Item 成功<br>武器之外 Item 消滅 | Monster 或 Player 死亡<br>Monster 死亡則消滅 | 無 |
| false | Player 使用 Item 失敗<br>**Player 不需強制移動** | 撤退成功<br>**Player 需強制移動** | Player 離開商店<br>**Player 需強制移動** |

表 1: 各 class 之 triggerEvent return value 所對應意義

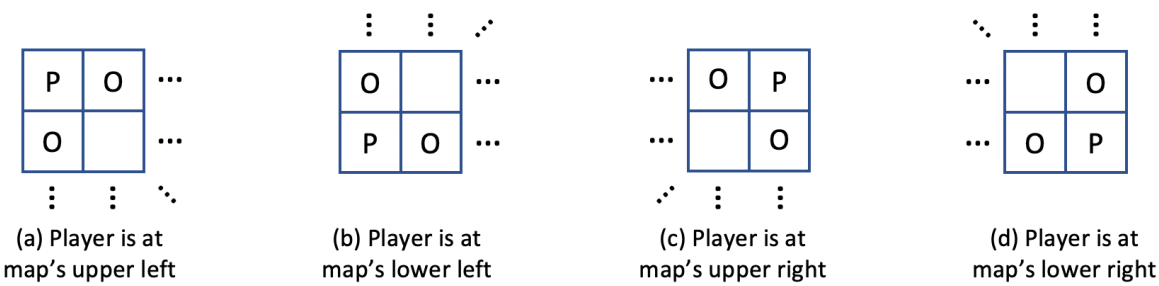


圖 4: 初期設計 Player 在地圖中遭遇之「移動卡死」示意圖，正方形代表一個 Room，P 代表 Player，O 代表物件

在滿血情況下，遭遇兩個 O 均為回血寶箱；因此，最後設定 1. 遭遇地圖寶箱一律不進行強制移動，2. 商店數量全地圖只設定一個，3. Boss 只有一個，4. Boss 除外的 Monster 在新遊戲開始時一定能被玩家擊倒。

**b-2. Showing Status：**

以 Player::triggerEvent(Object*)達成，沒有和其他物件互動因此傳入 null pointer，function 內先取得角色相關資訊，可透過直接取得 Player 屬性參數，和透過 getter 取得 GameCharacter、Object 屬性參數，最後用 cout 印出。

**b-3. Pick up Items：**

此功能對應三種場景：1. 玩家於迷宮移動時遇到寶箱道具，2. 玩家使用背包中的回血藥水，3. 玩家購買武器並裝備；三者的共通點即「使用 Item」，也就是執行 Item::triggerEvent(Object*)並將 Player pointer 作為參數傳入；表 1 提及，triggerEvent 返回值為 true 代表道具使用成功，換句話說，就是玩家角色能力數值成功改變，因此，這裡設計 Player::updateStatus(int, int, int, int)的返回值為 bool，並將其綁定在 triggerEvent 最後一行決定 triggerEvent 的返回值；在 updateStatus 中能將道具給予能力數值更新至玩家角色身上，例如血量、攻擊力、錢、爆擊率，這裡唯有血量更新可能造成 updateStatus 返回 false，目的是防止 GameCharacter::currentHealth 大於 GameCharacter::maxHealth，換句話說，角色在滿血情況下無法使用回血道具；程式中也設計了防止使用者重複裝備武器讓玩家角色攻擊力無限制疊加，在執行 updateStatus 前會先檢查 Player::weaponName 是否與道具名稱相同，相同則立即讓 triggerEvent 返回 false；在 updateStatus 中也設定了當傳入參數 money 大於 0 為獲得 money，小於 0 為花費 money。

第一種場景於 Dungeon::chooseAction()中進行，並設計讓玩家能夠遇到的寶箱有回血(GameCharacter::currentHealth)、增加額外攻擊力(Player::addedAttack)、和增加金錢(GameCharacter::money)三種，寶箱成功使用後會透過 Room::popObject()消滅。

第二種場景於 Player::useBackpack()中進行，遊戲設計玩家的背包只會有回血藥水；為了防止 **b-1.**提到的移動卡死，遊戲開始背包就放滿了回血藥水，而在遊戲進行中玩家可至商店購買藥水放入背包；在 useBackpack 中會先列出所有持有藥水供使用者選擇，若沒有藥水則直接退出；使用者選擇指定藥水後，可以選擇使用或丟棄，前者會執行 Item::triggerEvent，後者沒有任何動作，使用成功和確定丟棄後，均會照順序進行以下動作：1. 釋放道具所佔動態記憶體、2. 若道具不是在背包最後一格(vector 尾端)，後續道具均往前移動一個 vector 中的位置、3. pop_back()最尾端 vector element；雖然第 2 步大多情形為線性操作時間效率，但可以確保背包內藥水排序是按照獲得時間順序。

第三種場景發生於 NPC::triggerEvent(Object*)中的 NPC::buyCommodity(Player*, int)，玩家選擇購買武器後直接執行該武器的 Item::triggerEvent(Object*)，所以玩家很像與 NPC 對話後直接獲得能力，而非獲取道具在背包，離開商店後才裝備使用。

**b-4. Fighting System：**

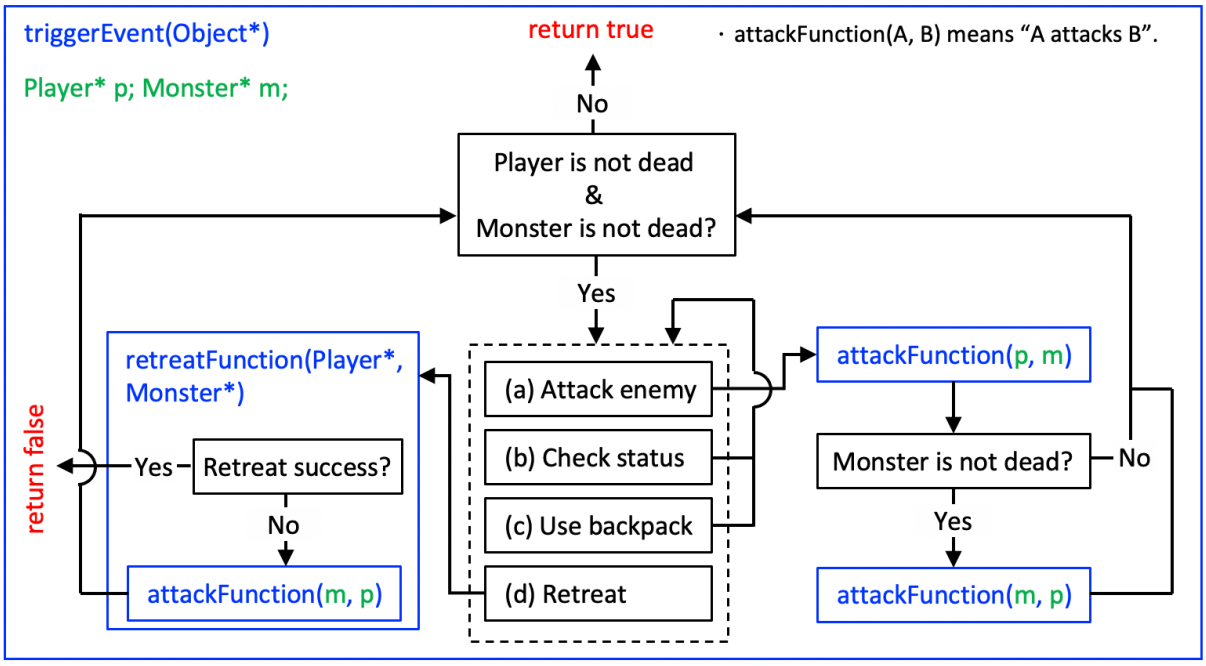


圖 5: Monster::triggerEvent(Object*)內部運作流程圖

這裡設計玩家一在房間遭遇 Monster 立即進入戰鬥，戰鬥於 Monster::triggerEvent(Object*)中進行，Player pointer 為傳入參數；圖 5 呈現玩家與敵人戰鬥的流程與用到之 Monster 操作方法，流程內容與各操作方法已於 a-4.說明，而 triggerEvent 的 return 值所對應的作用可再參照表 1；圖 5 中能在觀察到，本遊戲設計讓使用者能在戰鬥回合之間無限次確認狀態(Check status)和使用背包(Use backpack)。

## b-5. NPC：

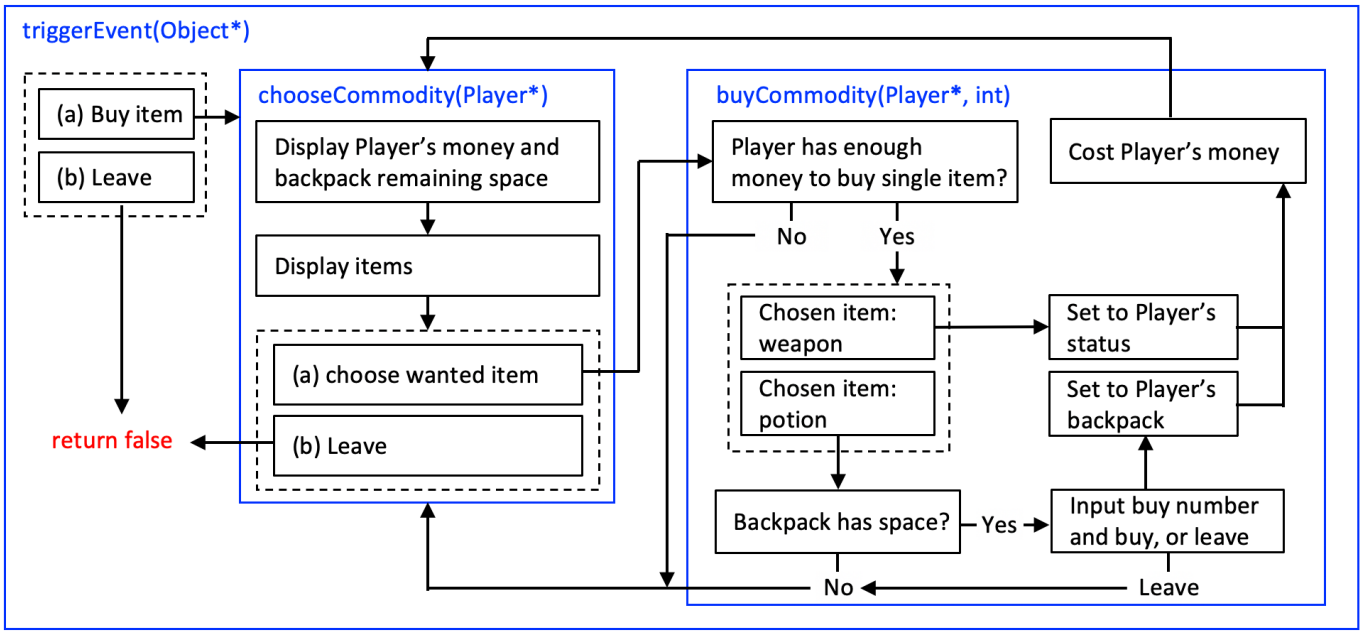


圖 6: NPC::triggerEvent(Object*)內部運作流程圖

同樣設計玩家一在房間遭遇立即進入商店，與商店互動於 NPC::triggerEvent(Object*)中進行，Player pointer 為傳入參數，與商店的互動流程圖如圖 6 所示；此 triggerEvent 只會回傳 false 值，發生在使用者選擇離開的時候，對應動作可再參照表 1；若選擇 buy item，會執行 chooseCommodity(Player*)，首先會呈現玩家角色所擁有的金錢與背包空間，再呈現商店所擁有的

道具；呈現道具資訊時會透過 Object::name 辨識道具是武器還是回血藥水並分別呈現對應資訊；使用者所輸入想購買的道具編號會作為傳入值傳入 buyCommodity(Player*, int)。

buyCommodity 會先確認玩家持有金錢足夠購買單一道具，在可以購買的情況下，武器道具會直接裝備在玩家上(即直接執行 Item::triggerEvent(Object*)，傳入值為 Player pointer)，而藥水道具會先檢查玩家背包是否有空間，在有空間的情況下才會讓使用者輸入想購買的藥水數量；程式也對使用者能輸入的數量做了限制，不可超過「背包空間數量」和「玩家持有金錢可購買最多藥水數量」兩者的最小值；輸入購買數量後，會 dynamic memory allocation 對應數量的 Item 並複製商品資訊，再把 Item pointer 存放在暫時的容器(vector<Item*>)，最後把容器傳入 Player::setBackpack(const vector<Item*>&)，即把輸入數量之藥水放入玩家背包；綜合上面說明，此遊戲設計並不會發生商品被購買完的情形；在 buyCommodity 最後，會執行 Player::updateStatus(int, int, int, int)傳入玩家應支付的金額，更新玩家所擁有的金錢；buyCommodity 執行完畢後，會再次執行 chooseCommodity，使用者可以選擇購買其他商品或選擇離開。

### b-6. Game Logic：

用來判斷遊戲是否繼續進行，若是判斷遊戲結束，則再判斷玩家輸贏，此功能透過 Dungeon::checkGameLogic()，如圖 3 所示；首先判斷玩家是否死亡，若死亡則遊戲結束且玩家輸了；若未死亡，再判斷玩家是否抵達終點，若未抵達，遊戲需繼續進行；若抵達終點，會再判斷魔王是否死亡，若未死亡，遊戲需繼續進行，若死亡，遊戲結束且玩家勝。

### b-7. Record System：

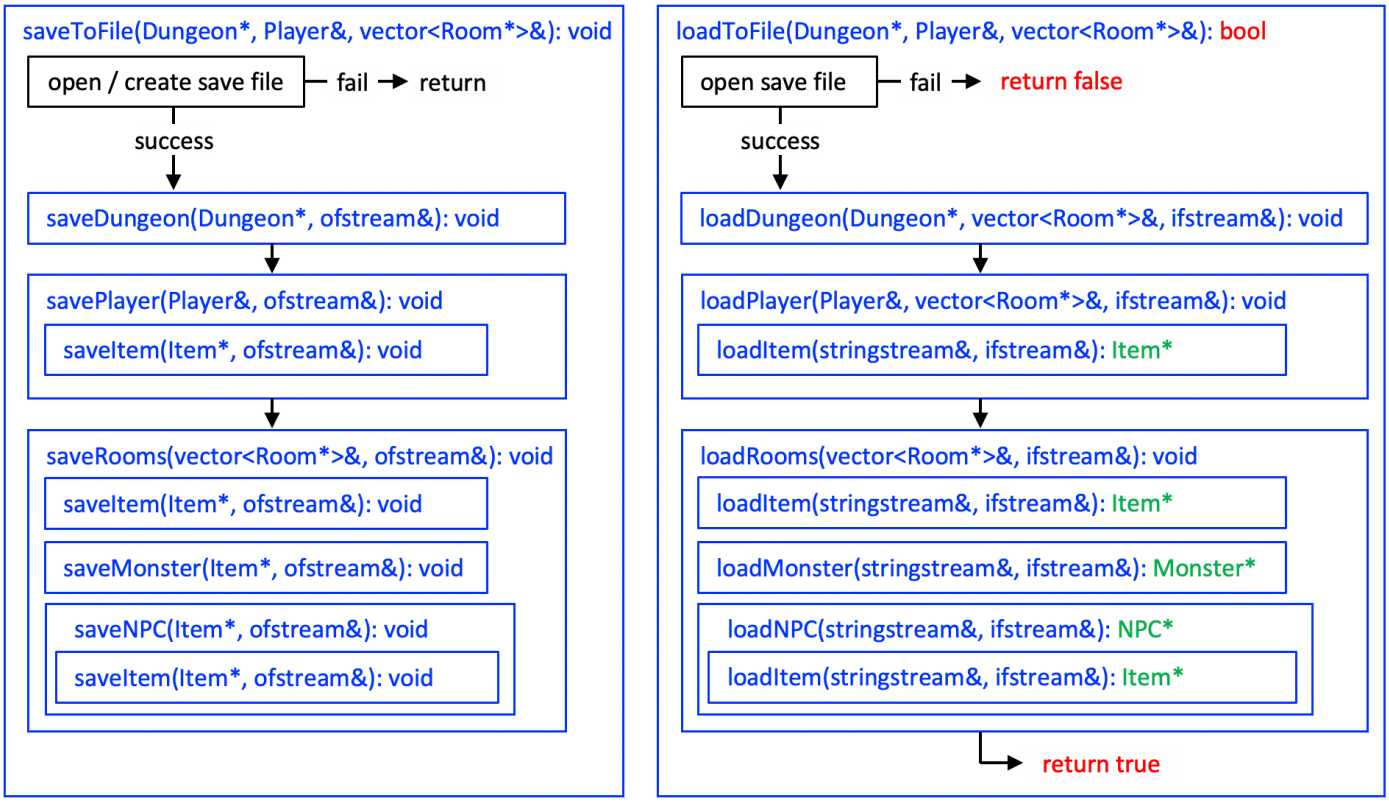


圖 7: 由 class Record 之操作方法所構成的存讀檔系統示意圖

遊戲存讀檔系統由 class Record 的操作方法組成，如圖 7 所示；saveToFile 用於存檔，無返回值；loadToFile 用於讀檔，返回值為 bool，用於開始選單當讀檔失敗時，使用者可以再選擇讀檔以外的選項；兩個操作方法首先都會檢查存讀檔用的 txt 文件是否能創造或開啟，成功了才會正式進行存讀檔;存檔先用 saveDungeon 記錄 class Dungeon 的一些屬性參數，再用 savePlayer 記錄 Player 的屬性參數；Player 被記錄的最後一個屬性參數為背包內道具數量，用於讀檔時決定迴圈生成道具的次數；存檔時背包不為空的情況下，會執行 saveItem 存下背包道具資訊；存檔最後用 saveRooms 記錄迷宮各房間屬性參數，值得注意的是，class Room 中是透過 Room pointer 取得相接的房間，然而 Room 在每次生成記憶體位置並非固定，因此存檔不能記錄記憶體位置，而是相接房間的編號；若沒有相接的房間，我們也會特別以「-1」記錄，避免記錄的資訊數量不一；房間中若有物件，我們會先以 dynamic_cast 識別 Object pointer，記下房間是否有 Monster、Item、NPC，若有，則分別用 saveMonster、saveItem、saveNPC 記錄資訊，而 saveNPC 中會再執行 saveItem 記錄商品；記錄時會將一個物件的資訊記在同一行，且每個資訊以間插入一個空格以 ofstream 輸出至 txt 文件，存檔結果如圖 8 所示。

savings.txt

```
1   Dungeon 4 5 3 3 6
2   Player YCLin 250 250 100 1000 10 15 15 Sword_lv.0 50 0 1 6 6
3   Item Potion_lv.0 100 0 0 0
4   Item Potion_lv.0 100 0 0 0
⋮
22  Room 6 0 0 1 11 5 7 1 0 0
23  Monster Boss 3000 3000 1000 1000 20
24  Room 7 0 0 2 12 6 8 0 1 0
25  Item HealthRecover_lv.1 95 0 0 0
26  Room 8 0 0 3 13 7 9 0 0 0
27  Room 9 0 0 4 14 8 -1 0 1 0
28  Item HealthRecover_lv.1 95 0 0 0
29  Room 10 0 0 5 15 -1 11 1 0 0
30  Monster Slime_lv.1 150 150 100 100 10
```

圖 8：執行存檔後遊戲資訊輸出至 txt 文件的結果

讀檔中每個物件生成或屬性參數更新的流程大致如下:1. ifstream 讀取 txt 檔案、2. 使用 getline 一次取一行 txt 文件資訊存放在一個 string、3. 將 string 輸入至 stringstream、4. 將 stringstream 輸出至對應數量與型態之臨時變數、5. 將臨時變數用 setter 設定於各 class 屬性參數或用於生成新物件輸入參數；若同一個 stringstream 變數會反覆使用，在執行新一次的 3.前，需對 stringstream 變數進行重置;由於 stringstream 輸出至變數是以空格作為 delimiter，Object::name 和 Object::tag 不可含有空格，否則會導致讀檔變數讀取會錯位。讀檔系統首先執行 loadDungeon，更新 class Dungeon 一些屬性參數並生成迷宮內所有空房間，需注意，由於本遊戲設計是能在遊戲中途讀檔，因此生成新空房間前要將 Dungeon::rooms 內所有舊房間清除；讀檔系統再來會執行 loadPlayer 更新 class Player 屬性參數，讀到 Player 含有道具時，會再執行 loadItem 生成道具存放置 Player 背包，同樣，由於能在遊戲中途讀檔，以迴圈執行 loadItem 前要將 Player 背包中舊的道具全部清空；讀檔系統最後執行 loadRooms，會對每個房間進行串接設定，並在讀取到含有 Monster, NPC 或 Item 時執行對應的 loadMonster, loadNPC 和 loadItem 生成新物件並置於房間中。

## (c) 額外功能實作細節：

### c-1. Show Map：

此功能為讓使用者查看玩家、怪物、魔王、寶箱、商店、終點於迷宮的位置，以 Dungeon::showMap()實現；由於迷宮設計為矩形，操作方法 showMap 中使用兩個迴圈遍歷所有房間，並使用 dynamic_cast 判斷房間是否含有怪物(含魔王)、寶箱、商店，使用 Room::getIsExit()判斷房間是否為終點，若有則將對應字母印出；而玩家位置在迴圈前就使用 Player::getCurrentRoom()和 Room::getIndex()取得所在房間編號，迴圈到對應房間時印出字母；若房間沒有任何物件或資訊則印出空白；資訊印出玩家的優先度最高，而剩下資訊在物件設置於迷宮時已確定彼此不會位置重複，這也是設計一個房間只含有一個物件的原因之一；另外，遊戲設計了在玩家未拜訪該房間的情況下，房間資訊不顯示在地圖上，為達到這項功能設計了屬性參數 Room::isVisited 在 showMap 的迴圈中使用；圖 9 呈現了此功能在遊戲運行時的呈現情形。

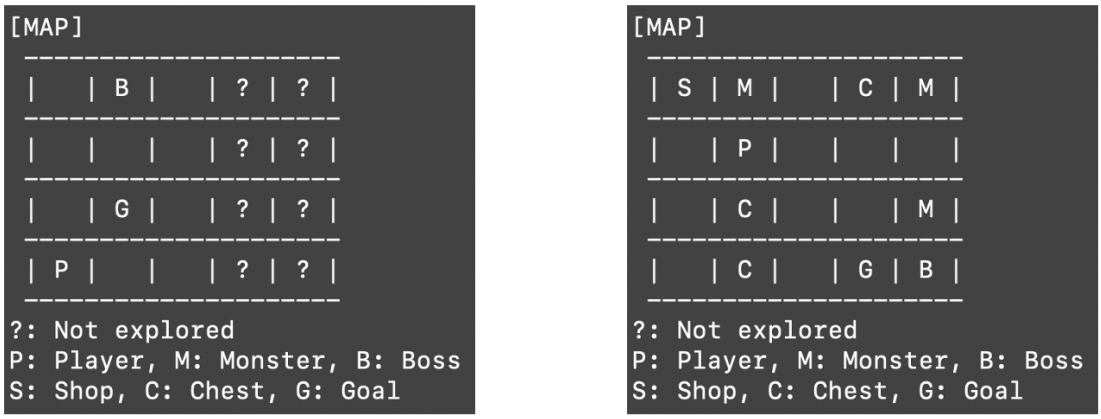


圖 9：Show Map 功能在遊戲中的呈現情形，左圖為迷宮尚未完全探索，右圖為迷宮完全探索完畢

### c-2. Random Function：

此功能用在產生隨機整數與事件機率計算，分成 randomInt(const int, const int): int 和 oddFunction(const int): bool，均定義在 Monster.cpp；使用隨機功能需加入#include <random>，亂數種子使用 random_device，亂數產生器使用 default_random_engine，兩者均以 global variable 定義在 Monster.cpp；randomInt 的兩個傳入參數設定了隨機整數產生的範圍，隨機整數的產生使用了 uniform_int_distribution (Reference: https://reurl.cc/zb97de)。

oddFunction 的傳入參數為機率大小設定，例如傳入 20，則有 20%機率 oddFunction 會返回 true，80%機率返回 false；原理為 oddFunction 內使用了 randomInt 產生一個 1 到 100 的隨機整數，再與傳入值比較是否小於等於，比較結果作為返回值。戰鬥系統中 Monster::attackFunction 加入了 oddFunction，在 attackFunction 中，當角色 A 攻擊角色 B 時，會先將角色 A 的爆擊率(GameCharacter::criticalAttackRate)傳入 oddFunction 來決定本次攻擊是否觸發爆擊，若觸發成功，便會把角色 B 承受到的傷害乘上 2 倍，若無則維持 1 倍，玩家角色的爆擊率可以透過購買武器改變，怪物則設定為固定值(Boss 的比較大)；而戰鬥撤退(Monster::retreatFunction)也加入了 oddFunction 來決定撤退成功與否。

Dungeon::getRandomRoomNumber()中也使用了 randomInt 產生隨機房間編號，即將 0, 迷宮房間總數–1 作為參數輸入 randomInt；getRandomRoomNumber 內會確保產生之隨機房間編

號符合：1. 不是玩家目前位置、2. 該房間沒有物件、3. 該房間不是終點，只要一項條件不合便會再重新產生隨機房間編號。

**c-3. Player Level System & Monster/Chest Generator：**

迷宮當下除了 Boss 之外的 Monster 數量，以 Dungeon::currentMonsterNumber 記錄，在 Dungeon::chooseAction 中若擊倒之 Monster 不是 Boss，便會使 currentMonsterNumber 減一，該值等於 0 時便會執行 Player::LevelUp()對玩家角色升級，當中會讓玩家等級(Player::level)加 1，最大血量(GameCharacter::maxHealth)增加 1 倍後回滿目前血量(GameCharacter::currentHealth)，還有基礎攻擊力增加 1 倍。

為了讓玩家能在迷宮中「練功」，Boss 之外的怪物與寶箱道具會自動生成，控制條件為：currentMonsterNumber 為 0 時便執行 1. Dungeon::createMonster()生成新 Monster，生成數等於 Dungoen::maxMonsterNumber，更新 currentMonsterNumber 等於 maxMonsterNumber、2. 當迷宮目前寶箱數量(Dungeon::currentChestNumber)小於迷宮允許最大寶箱數量(Dungeon::maxChestNumber)時，會執行 Dungeon::createChest()產生新寶箱補充，使目前數量等於最大數量；在 b-3.提到寶箱有三種，在 createChest 中也加入了 randomInt 來隨機決定是哪一種要被生成；另外，遊戲設定了 Monster 和 Chest 在生成時兩者的屬性參數會隨著玩家等級成長而成長。

**c-4. Input Optimization：**

由於遊戲的進行主要依賴使用者用鍵盤輸入選項，為了防止使用者有意或無意的不當輸入，設計了 inputOptimizer(const int, string): int 這個 function 在 Object.cpp 中，傳入第一個參數為選項的數量；假設傳入值為 4，使用者不是輸入數字 0 到 3 或字母 a 到 d(含大寫)則會要求重新輸入，是的話則返回選項對應整數；在處理鍵盤輸入時，只會將每次 input stream 中第一個非空白鍵 character 存入暫時變數，接著清空 input buffer，防止干擾到下一次選項選擇。

遊戲呈現重要資訊需要使用者注意時，螢幕會出現「Press Enter to continue ...」，使用者按下 enter 後遊戲才會繼續進行，此功能在 inputOptimizer 的第二個傳入參數為"pause"時提供，實作利用了 cin 在按下 enter 後才會執行下一行和 getline 可以將'/n'存入 string 的特性。

## III. Result

下面解說 demo 影片中不同時間點所進行的動作與對應實作功能：

**[00:00 - 00:04]:** 啟動遊戲 → 選擇新遊戲 → 輸入 Player 姓名

**[00:04 - 00:09]:** 顯示地圖 → 使用者選擇查看 Player 狀態 → 呈現 Player 資訊與地圖

✧ 展示功能: b-2. Showing Status, c-1. Show Map

**[00:10 - 00:37]:** 使用者操控 Player 移動 → Player 抵達終點房間 → 使用者操控 Player 移動 → 遭遇敵人進行戰鬥 → Player 使用背包藥水回血 → 擊敗敵人獲得金錢

✧ 展示功能: b-1. Movement, b-3. Pick up Items, b-4. Fighting System, b-6. Game Logic

**[00:39 - 00:42]:** 擊敗敵人獲得金錢 → 迷宮敵人(Boss 除外)消滅完畢，觸發 Player 升級 → 呈現 Player 資訊 → 進行存檔

✧ 展示功能: c-3. Player Level System & Monster/Chest Generator

**[00:47 - 01:05]**: Player 遇到回血型寶箱，因滿血無法使用 → 與敵人戰鬥造成損血 → 回去撿回血寶箱，順利回血

✧ 展示功能: b-3. Pick up Items, b-4. Fighting System

**[01:07 - 01:27]:** 進入商店 → 選擇購買武器並裝備至身上 → 再選擇購買藥水 → 離開商店 → 查看背包，多了在商店購買的藥水

✧ 展示功能: b-3. Pick up Items, b-5. NPC

**[01:29 - 01:36]:** 擊敗敵人獲得金錢 → 迷宮敵人(Boss 除外)消滅完畢，觸發 Player 升級，與 01:25 畫面比較明顯看到新的 Monster 和 Chest 生成 → 呈現 Player 資訊 → 進行存檔 → Player 移動到新位置，獲得金錢寶箱

✧ 展示功能: b-3. Pick up Items, c-3. Player Level System & Monster/Chest Generator

**[01:37 - 01:51]:** 進行存檔 → 與 Boss 進行戰鬥並將其擊敗

✧ 展示功能: b-7. Record System

**[01:53 – 02:14]:** Player 移動至敵人所在位置觸發戰鬥 → 使用者選擇撤退，成功 → Player 回到上一步位置 → 再次移動 Player 至敵人所在位置觸法戰鬥 → 使用者選擇撤退，失敗 → 敵人進行攻擊，Player 死亡，遊戲輸了 → 遊戲結束

✧ 展示功能: b-6. Game Logic, b-7. Fighting System(retreat action), c-2. Random Function

**[02:14 – 02:35]:** 進行讀檔，回到與 Boss 對戰前 → 與 Boss 對戰，Player 觸發爆擊，一刀擊敗 Boss → 移動 Player 至終點，遊戲贏了 → 遊戲結束

✧ 展示功能: b-6. Game Logic , b-7. Record System, c-2. Random Function

**[02:37 – 02:46]:**

✧ 展示功能: c-4. Input Optimization

## IV. Conclusion

本作業以 C++成功實作了文字冒險遊戲，遊戲關鍵物件 Player, Monster, NPC, Item, Room 使用了物件導向的概念構成，Player 與 Monster, NPC, Item 間的互動使用了 virtual function 建構，而存檔系統使用了 File I/O 的方法；最後，所有物件包含至 class Dungeon 並透過其多個操作方法組織遊戲的進行；報告中先說明了遊戲運行流程，再詳細說明了程式實作細節，最後影片展示了作業要求的所有基本功能：Movement, Showing Status, Pick up Items, Fighting System, NPC, Game Logic, Record System，也展示了額外增強功能：Show Map, Random Function, Player Level System & Monster/Chest Generator, Input Optimization。