**I. Introduction**

本程式為以C++語言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**

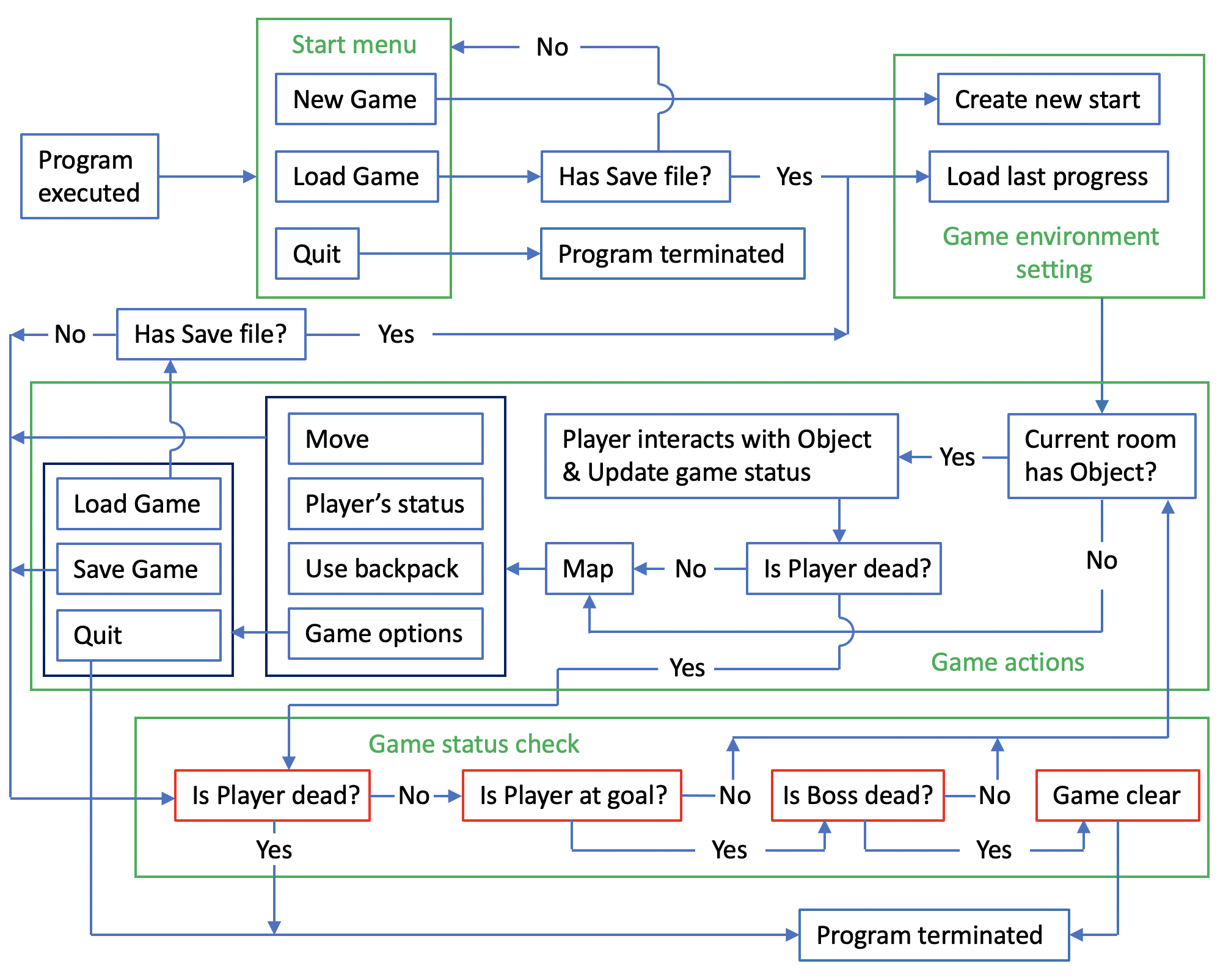


圖1: 程式運作流程圖，箭頭指向代表程式進行方向

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。

一張含有 文字 的圖片

自動產生的描述

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

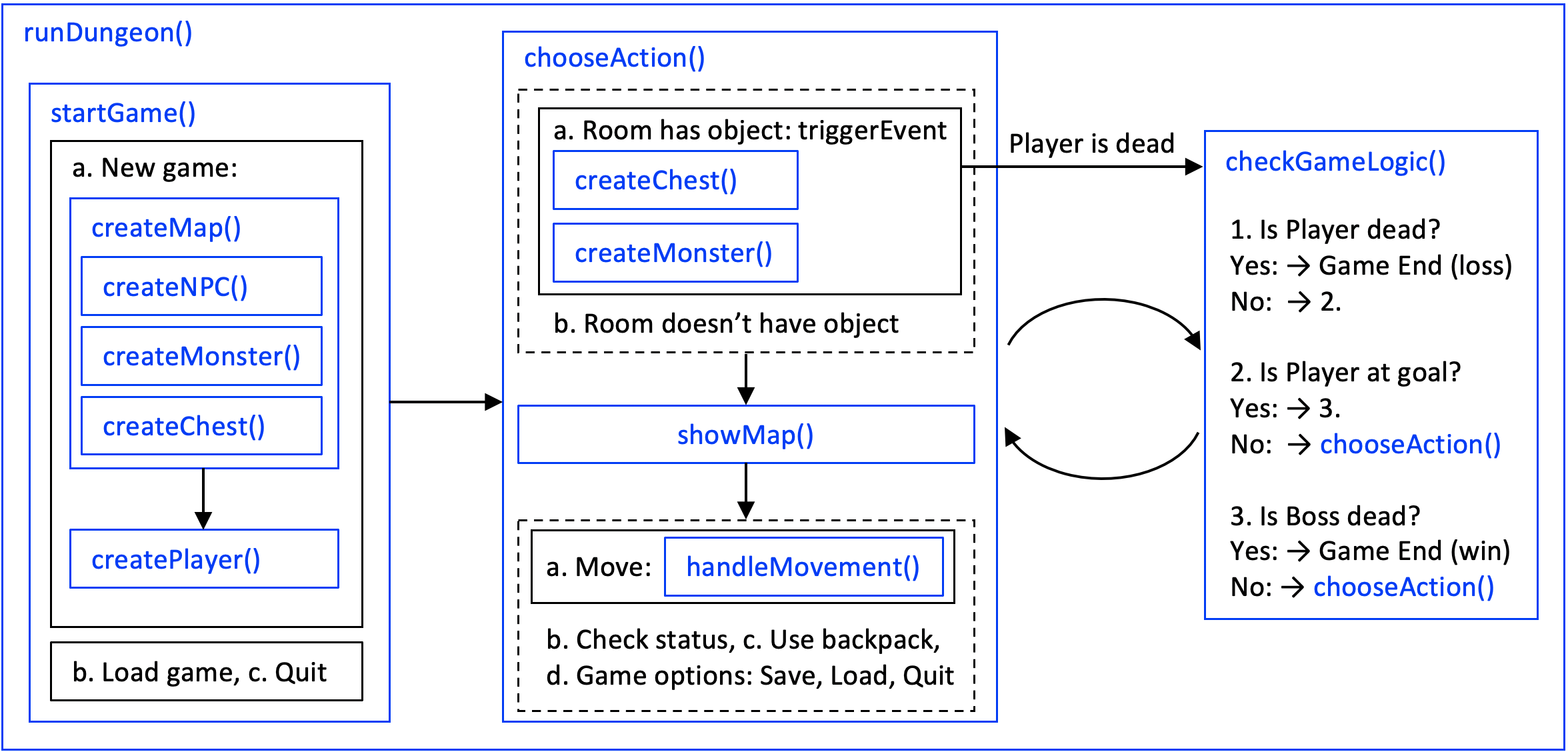


圖3: 透過class Dungeon之操作方法所構成之遊戲運行示意圖

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，在destructor中均執行了release memory。

在header file有互相include的情況下，如Player.h & Item.h和Dungeon.h & 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：**

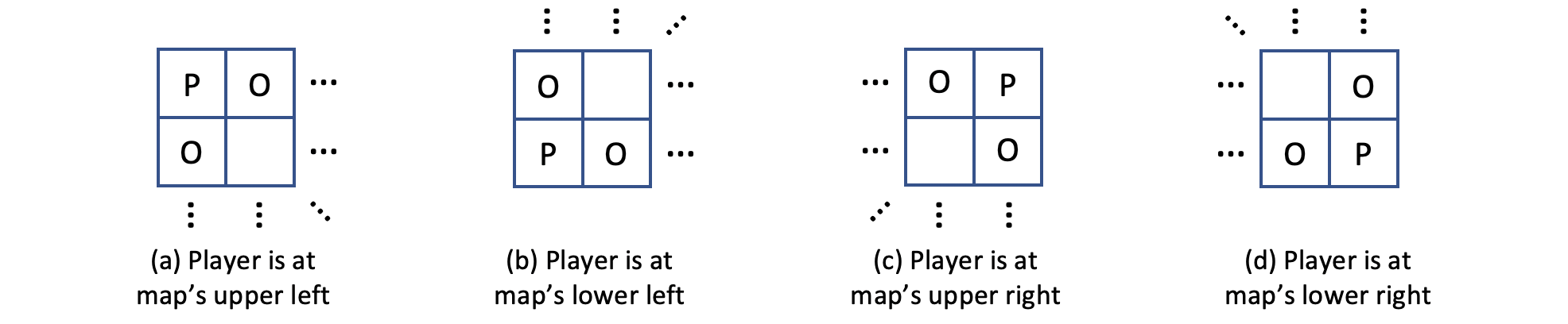
**b-1. Movement：**

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，作業要求撤退功能需要將玩家強制移動至前一次移動房間，本遊戲也設定了讓Player離開商店會強制移動至前一次移動房間，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無法離開地圖角落的問題，或是Player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

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

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：**

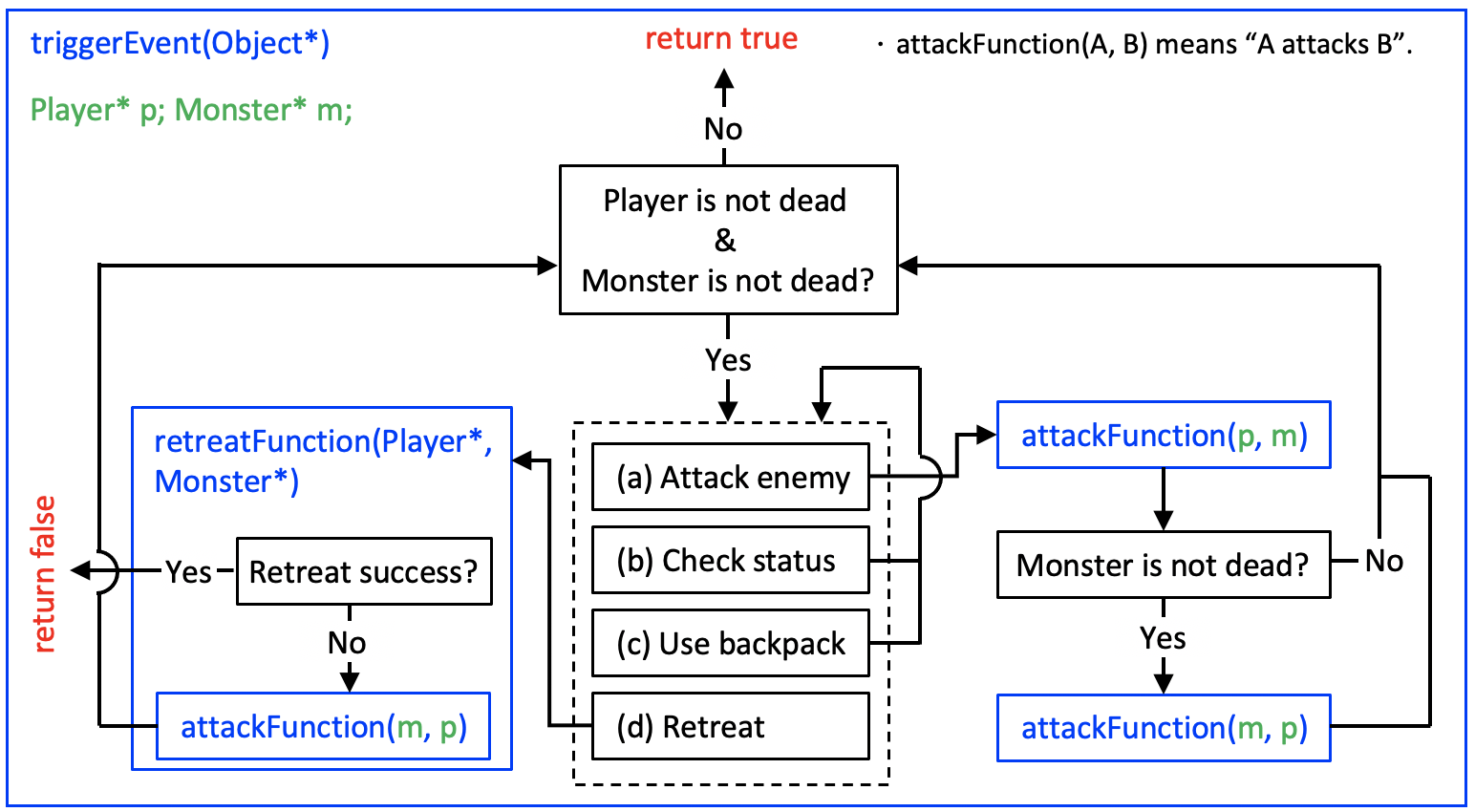


圖5: Monster::triggerEvent(Object\*)內部運作流程圖

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：**

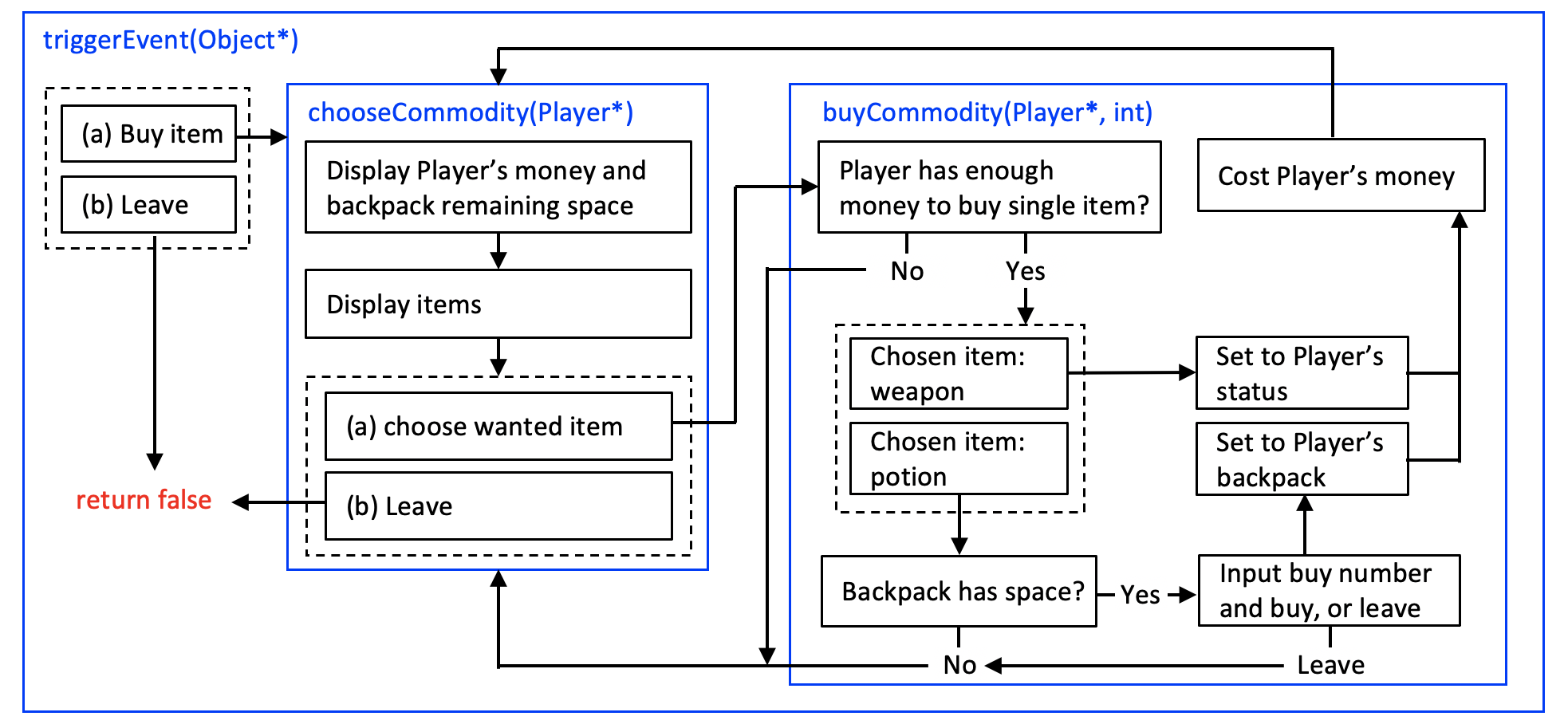


圖6: NPC::triggerEvent(Object\*)內部運作流程圖

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：**

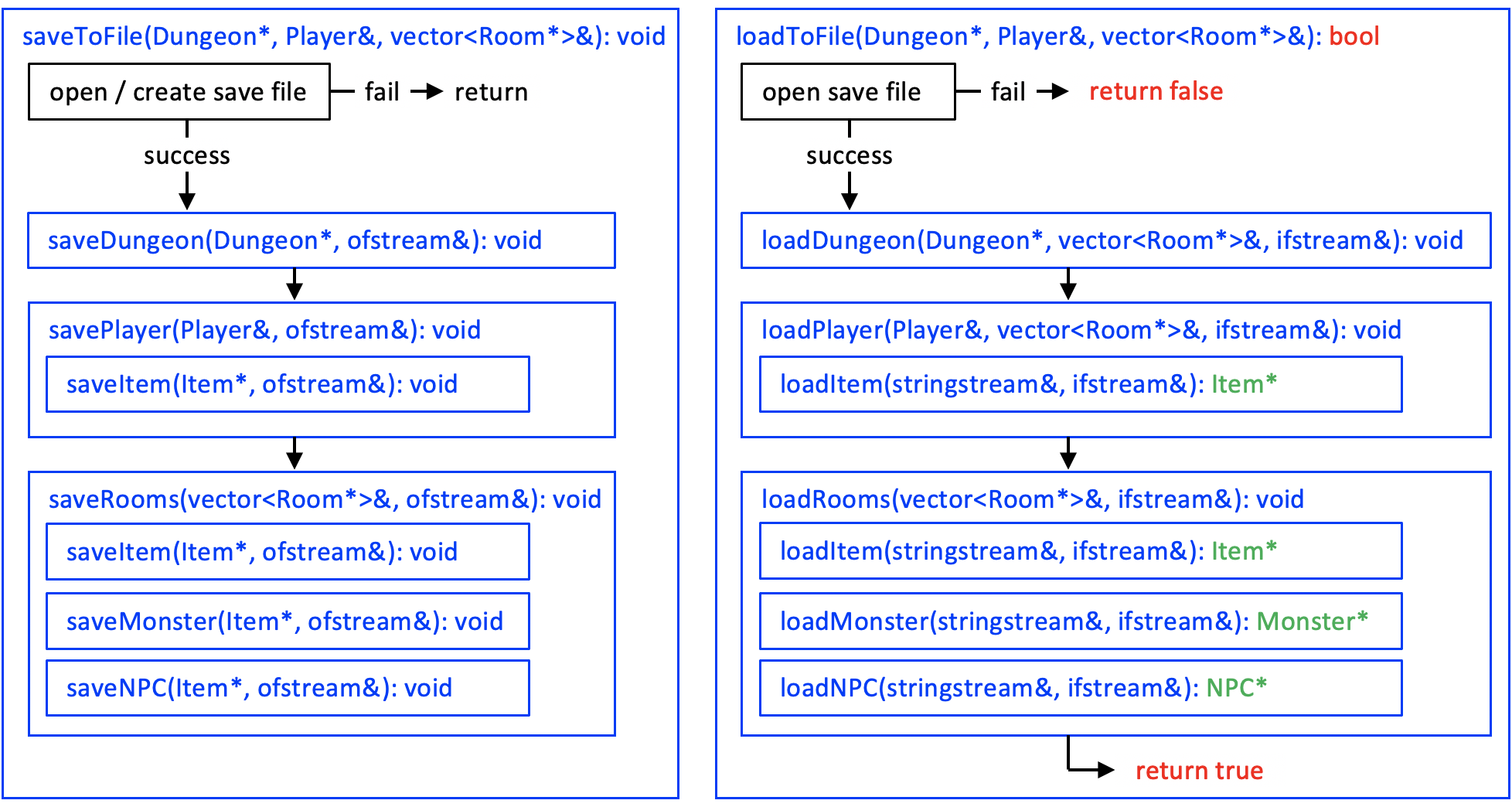


圖7: 由class Record之操作方法所構成的存讀檔系統示意圖

；Player額外獲得的攻擊力記錄在屬性參數weaponAttack和addedAttack，細節如下：

* weaponAttack: 裝備武器獲得的攻擊力
* addedAttack: 撿寶箱道具獲得的攻擊力 + weaponAttack
* GameCharacter::attack: 玩家基礎攻擊力 + addedAttack + weaponAttack

- Basic Functions

- Optional Enhancement

Results

Discussion

Conclusion