**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**

**1. 程式執行概要**

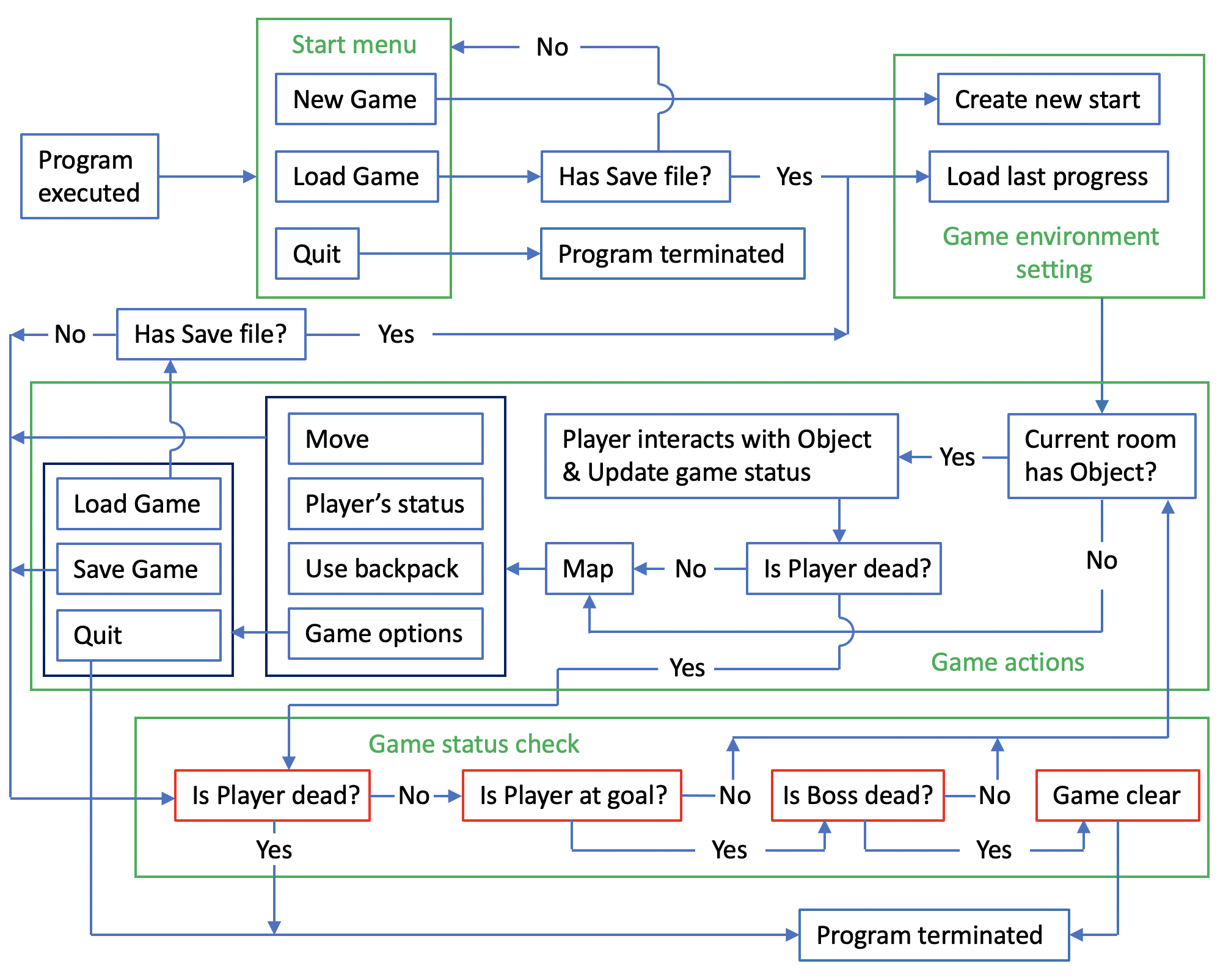


圖1: 程式運作流程圖，箭頭指向代表程式進行方向

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實作細節**，**(b) 基本功能實作細節**，**(c) 額外功能實作細節**。

**(a) 各class實作細節：**

**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(attribute)：Object::name和Object::tag，後續繼承Object的class可以再另外設定Object::name，例如玩家的名字；Object::tag雖然在遊戲進行中不會顯示，但在存讀檔上有很大作用，可透過辨識Object::tag對不同物件進行對應存讀檔；Object的操作(operation): Object::triggerEvent設定為pure virtual function，能讓繼承Object的class各自去override自己的triggerEvent並定義要從事的行為，程式執行中可透過late binding自動呼叫對應class的triggerEvent。

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

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::triggerEvent內定義了玩家使用道具後，如何提升玩家角色的能力值—血量、攻擊力、錢、爆擊率，而以上四項也就作為了Item的屬性參數(Item::health, Item::attack, Item::money, Item::criticalAttackRate)；本遊戲中生成的Item分成三種：寶箱道具、回復藥水、武器，在Item::triggerEvent中透過辨識Object::name來分別處理與玩家的互動(line 7, 20, 22 in Item.cpp)。

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的屬性和操作主要為玩家和怪物的共同屬性和操作，屬性包含：最大血量(GameCharacter::maxHealth)、目前血量(GameCharacter::currentHealth)、攻擊力(GameCharacter::attack)、錢(GameCharacter::money)、爆擊率(GameCharacter::criticalAttackRate)；操作包含兩項：1. 決定角色是否死亡(GameCharacter::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(GameCharacter::takeDamage)，將對手的攻擊值作為argument傳入，減少目前血量。

由於Object和GameCharacter設計為abstract class，因此Object的triggerEvent為pure virtual function，GameCharacter不用override triggerEvent；須注意在程式編寫上，不可將Object和GameCharacter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，因此屬性中須含有Room pointer來指向被串接的Room，本遊戲設計每一個Room實體最多可以連接額外四個Room，分別在上下左右四個方位(Room::upRoom, Room::downRoom, Room::leftRoom, Room::rightRoom)；本遊戲設計一個房間最多含有一個道具或遊戲角色，因此以一個Object pointer (Room:: 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Room::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(Room::index)解決；遊戲通關條件之一為玩家抵達終點Room，因此透過Room::isExit決定此Room是否為終點，而Room::isVisited能決定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Room pointer參數Player::currentRoom和Player::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Player::changeRoom，將移動目的地房間之address作為argument傳入，將Player::currentRoom覆蓋Player::previousRoom，然後目的地房間address覆蓋Player::currentRoom。

第二個功能是讓Player能透過撿寶箱道具或裝備武器來提升攻擊力，並將額外獲得的攻擊力記錄在Player::weaponAttack和Player::addedAttack，細節如下：

Player::weaponAttack: 裝備武器獲得的攻擊力

Player::addedAttack: 撿寶箱道具獲得的攻擊力 + Player::weaponAttack

GameCharacter::attack: 玩家基礎攻擊力 + Player::addedAttack + Player::weaponAttack

- Basic Functions

- Optional Enhancement

Results

Discussion

Conclusion