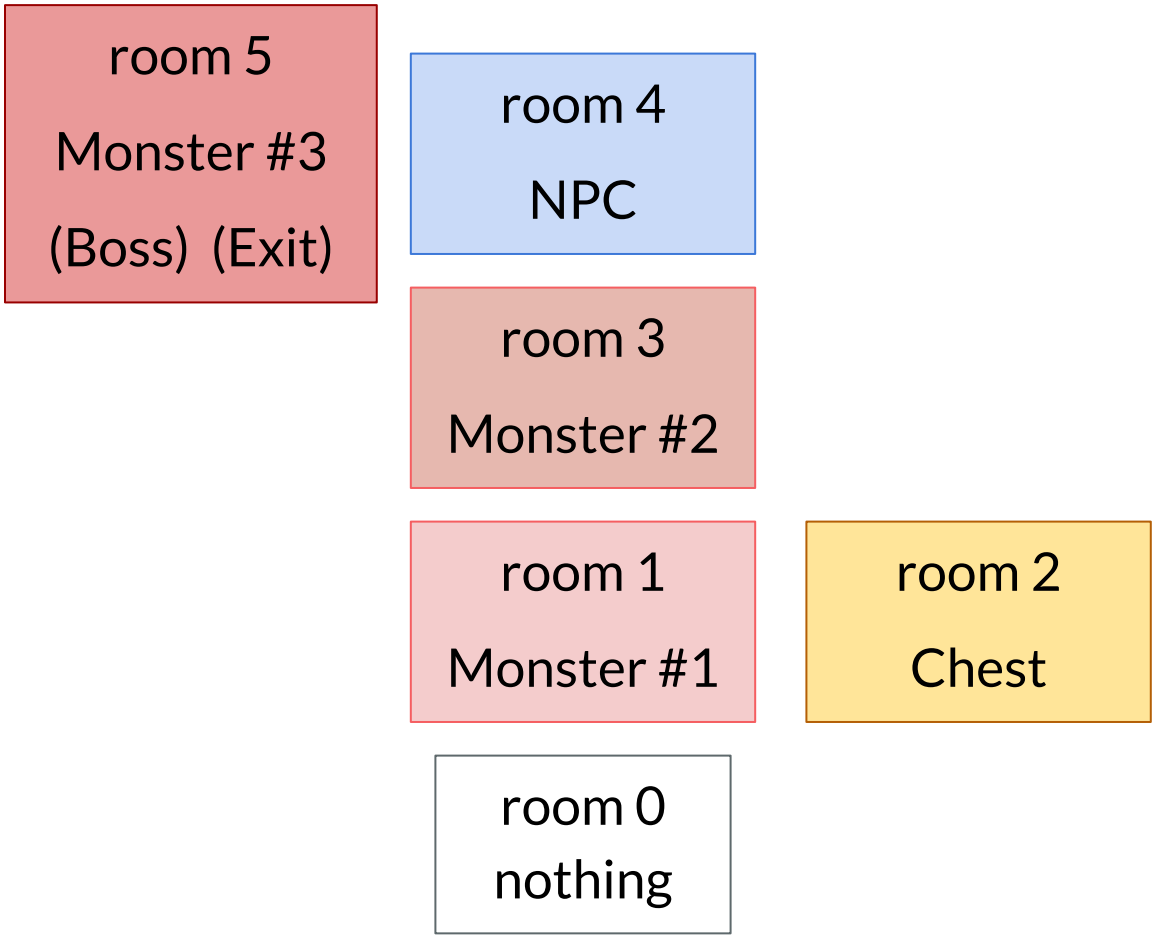
Dungeon
report

109550064

Game map

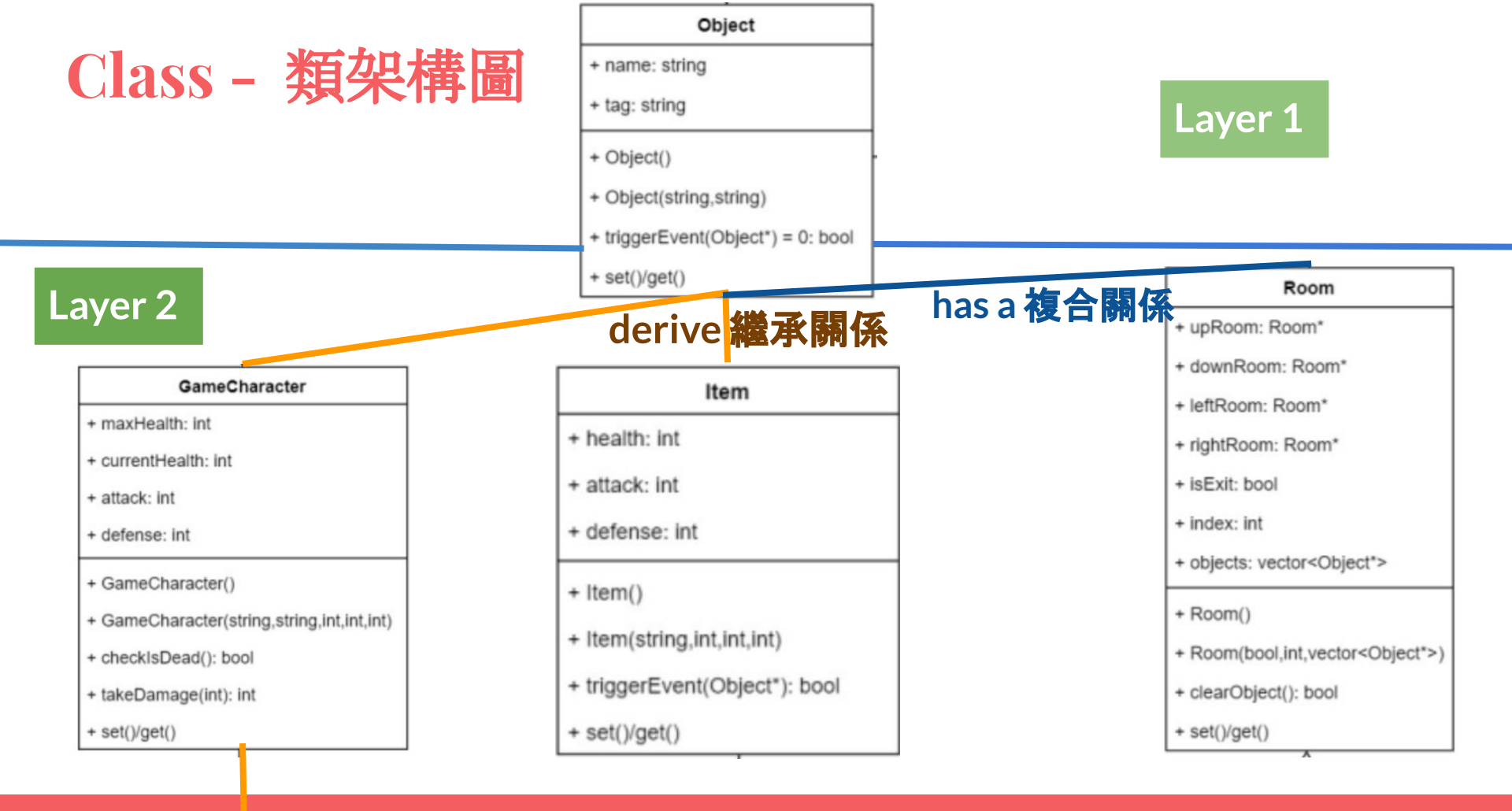

room 5
Monster #3
(Boss) (Exit)
room 4
NPC
room 3
Monster #2
room 1
Monster #1
room 2
Chest
room 0
nothing

report 製作順序 - 依照建構世界的順序

0. 各種class的架構
1. class的初始化及constructor, 簡單的 get/set 的 function 實作及注意事項
2. trigger event 的 virtual function & inheritance 的實現

   及 Player, Item, Monster, NPC 的 triggerevent 在幹嘛~
3. Movement 描述
4. Game Logic
5. Record system
6. 加分 - 配樂

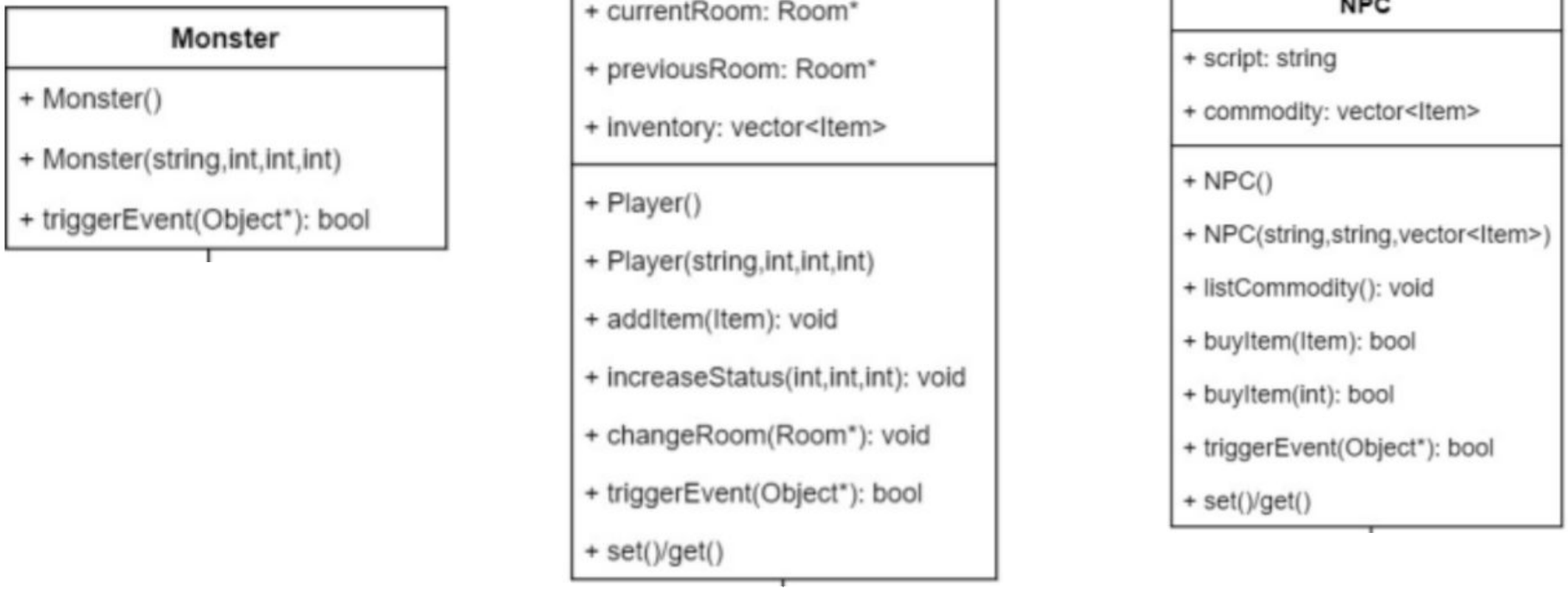
Class - 類架構圖


Object
+ name: string
+ tag: string
+ Object()
+ Object(string,string)
+ triggerEvent(Object*) = 0: bool
+ set()/get()
Layer 1
Layer 2
derive 繼承關係
has a 複合關係
GameCharacter
+ maxHealth: int
+ currentHealth: int
+ attack: int
+ defense: int
+ GameCharacter()
+ GameCharacter(string,string,int,int,int)
+ checkIsDead(): bool
+ takeDamage(int): int
+ set()/get()
Item
+ health: int
+ attack: int
+ defense: int
+ Item()
+ Item(string,int,int,int)
+ triggerEvent(Object*): bool
+ set()/get()
Room
+ upRoom: Room*
+ downRoom: Room*
+ leftRoom: Room*
+ rightRoom: Room*
+ isExit: bool
+ index: int
+ objects: vector<Object*>
+ Room()
+ Room(bool,int,vector<Object*>)
+ clearObject(): bool
+ set()/get()

Layer 3

GameCharacter (Layer 2)

derive 繼承關係



Layer 4

Player (Layer 3)

Room (Layer 2)

has a 複合關係
(不是繼承關係..

btw,

object 的 **tag** 屬性在此程式中扮有非常非常重要的地位，很多時候都需要藉此判斷其是哪個 derived class 才能對他們進行相對應操作，是助教們設計程式的小巧思~~

class 圖架構完了就來寫各個 constructor跟簡單的get/set 函式八~~

Constructor

☆ derived class 的 constructor 一次只能往 base call 一層

Q: 如何實現 Layer3 對 Layer1 的 private member 的初始化?~

Ans: GameCharacter(L2)其實沒有實際被使用者call 到的機會，都是被它 L3 的 derived class call，所以可以達到變數傳遞(人工寫)的功能~

"把 L1 需要的東西由 L3 傳給 L2，Let L2 可對 L1 初始化"

```
GameCharacter::GameCharacter(string name, string d_tag, int maxHealth, int attack, int defense)
:Object(name, d_tag) //** tag is d_tag
{
```

```
Monster::Monster(string name, int maxHealth, int attack, int defense)
:GameCharacter(name, "Monster", maxHealth, attack, defense) {}
//, Object(name, "Monster"): {}  //****多層繼承建構子寫法
```

constructor - member initializer list

```
Room::Room(bool isExit, int index, vector<Object*> objects)
:isExit(isExit), index(index), objects(objects) //member initializes list ** only can use on constructor
{
    upRoom = downRoom = leftRoom = rightRoom = NULL;
}
```

可省去寫 this pointer 的 effort (同名變數不用加 this)

非 constructor 要設父類別的 private value 的實現

```
void Player::increaseStates(int maxHealth,int attack,int defense)  //**升級嗎~
{
    //不能用this去access父類別的private member
    //***基類的private member只能被基類的友元函式或基類的成員函式access
    //用GC的member function 去 reset 值
    GameCharacter::setMaxHealth(maxHealth);
    GameCharacter::setCurrentHealth(maxHealth);
    GameCharacter::setAttack(attack);
    GameCharacter::setDefense(defense);
};
```

遍歷 vector 的方法 - NPC :: listCommodity()

1. 跌代器 (現在少用了
2. 把 vector 當陣列，用 index 0 ~ size-1 去遍歷

   (vec名.size -> 可 get 此 vector 內目前持有的元素個數，是 vector 的內部 function)

3. auto x

   for(auto x:this->commodity) {...} - 在 commodity 內的每個元素

複合關係

```
vector<Object*> objects; /*contain 1 or multiple objects, including monster, npc, etc*/
                         //複合關係: Room內有object指針
```

Item 的 << operator 重載 (程式層級)

☆ 讓每次輸出 item 類 object 時能夠直接輸出所有屬性，節省精力重新呼叫多個getter

Item 的 .h 檔先宣告（不然會報錯）> `ostream& operator<<(ostream&, Item&);`

.cpp 實現 v

```
//**程式層級的operator overloading for cout<<Item
ostream& operator<<(ostream& out, Item& k) //const for 不希望改到值, reference for 不需要copy
{
    string name;
    name = k.getName(); //get name is the public function 2 layers before
    int health, attack, defense;
    health = k.getHealth();  //**k: just like the object itself, can be used as the obj
    attack = k.getAttack();
    defense = k.getDefense();

    out<<"name: "<<name<<"\n"<<"hp: "<<health<<"\n"<<"attack: "<<attack<<"\n"<<"defense: "<<defense<<"\n";
    return out;
}
```

使用時可直接 cout item 型態 object (x)~~

```
for( auto x:this->commodity )   /
    cout<<x;   //用運算符重載實現
```

2. 各角色的 trigger event

Overall

Item, Monster, Player, NPC 都有從 Object 那邊繼承 triggerevent，為了使之後可以實現程式多態，所以 object 那邊將此 function 宣告為 (pure) virtual function

各自的功能則是在 derived class 內各自實現

```
/* pure virtual function */
virtual bool triggerEvent(Object*) = 0;   // = 0 表示是純虛函數，在此層不用寫要實現的東西
```

程式運行的部分，是從 Dungeon.cpp 看起，Dungeon 內會有一個 player，在遊戲中如果玩家選擇要與此房間內的角色進行互動，那就會用一個 for 迴圈去遍歷此房間的 object* vector，把 player 當成參數傳進每個 Object* 指向的 trigger event function(程式多態呈現處)，而每個 Object* (基類)內會依據所儲存的 "派生類指針"，指向 "各個派生類" 自己所實現的不同的triggerevent，而達成讓 player 與不同的角色進行相對應的互動的功能

```
void Dungeon::handleEvent(Object* pp)  // 傳入的變數是 Player 的 pointer
{
    // 1. find the "room" that the player in first
    // 2. get all the object in the room then ( use the get function in Room then
    // 3. use the objects' trigger event function and pass the player pointer in

    //pp need to down cast first
    Player *p = dynamic_cast<Player*>(pp);
    Room *cr = p->getCurrentRoom();
    vector<Object*> obj = cr->getObjects();

    //told the player what's in this "room"

    for( auto x : obj )
    {
        cout<<"There is a/an "<<x->getTag()<<" \""<<x->getName()<<"\" in this room."<<endl;
        x -> triggerEvent(p);
    }
        // 其實所有傳進來的變數型態都是player，但在後面的func 中對player值進行操作時，都要dow

};
```

另外很神奇的設計是，handle event 接收的變數型態都是 Object*，所以要把 player 的 pointer 當成 Player* 型態丟給 triggerevent前都要先down_cast一次

Player - show status

```
bool triggerEvent(Object*);
```

Derived class 這邊的宣告前面不用再加 virtual 關鍵字了

```
bool Player::triggerEvent( Object* pp )
{
    Player *p = dynamic_cast<Player *> (pp);
    cout<<"["<<this->getName()<<"]"<<endl   //[Player's name]
    <<"> Health: "<<this->getCurrentHealth()<<"/"<<this->getMaxHealth()<<endl
    <<"> Attack: "<<this->getAttack()<<endl   //> Attack: 30
    <<"> Defense: "<<this->getDefense()<<endl;   //> Defense: 0

    return true;
}
```

就是簡單的 cout，一樣 player 傳進來的時候要線 down cast 一次，才能指到自己才有而 Object 沒有的函式 ex: getCurrentHealth(), getMaxHealth(), getAttack(), getDefense(), (但我剛剛發現這邊直接寫用 this 去指了哈哈

因為這些 function 不是 virtual function, 必須用準確的層級才能調用到自己的function

Item - open the chest

```
if( this->getName() == "Chest" )
{
    cout<<"You have opened the chest, and";
    p->addItem( (*this) );  //**
    p->getCurrentRoom()->eraseObject(0);
}
```

```
void Player::increaseStates(int hp,int ap,int dp)
{
    //不能用this去access父類別的private member
    //***基類的private member只能被基類的友元函式或
    //用GC的member function 去 reset 值
    setMaxHealth(this->getMaxHealth()+hp);
    setCurrentHealth(this->getCurrentHealth()+hp);
    setAttack(this->getAttack()+ap);
    setDefense(this->getDefense()+dp);
};
```

```
void Player::addItem(Item i) //**add item into the vector
{
    this->inventory.push_back(i);
    increaseStates(i.getHealth(), i.getAttack(), i.getDefense());
};
```

pickup item 主要是在 NPC 內實現，我們留待後面解釋NPC 部分的 pickup item

那 Open the Chest 的部分勒，就是：

1. 先調用 player 自己的 add item function，把東西放進包包並increase status

2. 把寶箱從房間內移除 (因為已經開過了)

NPC – show the script & trading

- Show the script

NPC's script:

Hi, Potter! Finally see you ha!

Here's all the things I have.

Hope these can to help you to defeat Voldemort, Good luck...

```
bool NPC::triggerEvent( Object* pp )
{
    Player *p = dynamic_cast<Player *> (pp);

    cout<<getScript();
    if(this->commodity.empty())
    {
        cout<<"\nI don't have any item now~ QQ\n";
        return true;
    }

    this->listCommodity();
```

就 cout script 阿

然後如果 NPC 已經沒東西了那就直接 return 了不用 list commodity

Trading

```
this->listCommodity();

cout<<"Please enter the number of the commodity you would like to bu
int n; cin>>n;
cout<<"\n"; //**not sure here
if( n >= this->commodity.size() )
{
    cout<<"\n"<<"enter invalid\n";
    return true;
}

cout<<"Item "<<this->commodity[n].getName()<<" is added to the backpack, and";
p->addItem(this->commodity[n]);  //**
cout<<" your status is updated: \n"<<endl;
p->triggerEvent(p);  //show the player's status

this->commodity.erase(commodity.begin() + n);  //**刪除陣列中某元素
}
```

```
void NPC::listCommodity()
{
    int i = 0;
    for( auto x:this->commodity )
    {
        cout<<"\n"<<i<<".\n"<<x;
        i++;
    }
}
```

NPC 先 list 自己的 commodity 給 player 選要買的東西，player 輸入要的 item 的編號

檢查編號有沒有越界，沒有的話繼續執行 additem 的動作並 show player 的 new status （同item那邊的描述

最後記得把 item 從 NPC 的 commodity list 刪除　　< 這樣就完成 Trade 了~~>

Monster - fighting system

```
bool Monster::triggerEvent( Object* pp )  //combat system included
{
    Player *p = dynamic_cast<Player *> (pp);

    cout<<"cuz there is a monster room, you can only choose: \n"<<endl
    <<"A. stay and fight the monster"<<endl
    <<"B. retreat to the previous room"<<endl
    <<"貼心小提醒: 沒打敗怪物不能前進喔 ^^"<<endl;
```

先輸出必須打怪才能前進的提示語，給player 選 attack 還 retreat

retreat 的話會直接退回先前房間

```
else if( di == 'B' )
{
    p->changeRoom( p->getPreviousRoom() );  // handle retreat in monster
}
```

Attack

會先輸出怪物資訊再進入打怪迴圈~~

```
while(1)
{
    //player attack first
    this->takeDamage( p->getAttack() - this->getDefense() ); //p's attack - m's defense
    cout<<"You hit Monster "<<this->getName()<<" for "<<p->getAttack()<<"points!"<<endl;
    if( this->checkIsDead() )
        break;

    //monster attack next
    p->takeDamage( this->getAttack() - p->getDefense() ); //m's attack - p's defense
    cout<<"Monster "<<this->getName()<<" hit you for "<<this->getAttack()<<"points!"<<endl;
    if( p->checkIsDead() )
        break;

    while(1)
    {
        int bol = attack_retreat();
        if( bol == 1 )
            break;
        else if( bol == 0 )  //retreat
        {
            p->changeRoom( p->getPreviousRoom() );
            return true; //handle 完就 retreat
        }
        else
        {
            cout<<"enter invalid!\n";  //continue
        }
    }
}
```

戰鬥中

1. 輪流調用 ”一起從GameCharacter那邊繼承”的 takeDamage 進行"被"攻擊的動作，每"被"攻擊一次就要確認該腳色是否存活

   player die - 遊戲結束　　monster die - 稍後敘述

2. 每輪互相攻擊結束後可選擇下回合要攻擊還是撤退，調用 attack_retreat() 函示完成這步

take Damage

```
int GameCharacter::takeDamage( int d )
{
    this->currentHealth -= d; // 操作放
    return this->currentHealth;
}
```

傳進來的 d 已經是

敵人的 attack - 自己的 defense 了

Check_is_dead（也是從 GameCharacter 那邊繼承的

```
bool GameCharacter::checkIsDead()
{
    if( this->currentHealth <= 0 )
        return true; //is dead
    else
        return false;
}
```

如果其中一方死亡，直接跳出 Monster 內的攻擊迴圈，等待遊戲主迴圈對遊戲邏輯進行判斷

```
int attack_retreat()
{
    cout<<"\nChoose next action:\n"
    <<"A. Attack!\n"
    <<"B. Retreat!\n";
    char dii;
    cin>>dii;

    if( dii == 'A' )
        return 1; //1 means continue
    else if( dii == 'B' )
        return 0; //0 means retreat
    else
        return 2;  //invalid input

}
```

Attack_Retreat()

讓 player 選要不要繼續攻擊，依照返回值在 Monster 的迴圈判斷要不要

break 出攻擊迴圈 +

送 player 回去前一個 room

(用 changeRoom() + getPreviousRoom() 實現)

3. Movement

```
void Dungeon::handleMovement() // all things about mo
{                              //外部迴圈怎麼呼叫是他們的事，自
    char di;

    if( checkMonsterRoom() )
    {
        handleEvent(&player);
    }
    else  //not moonster room
    {
        cout<<"Where do you want to go?"<<endl
        <<"A. Go up"<<endl
        <<"B. Go down"<<endl
        <<"C. Go left"<<endl
        <<"D. Go right"<<endl
        <<"E. Cancel"<<endl;
        cin>>di;
        cout<<"\n\n";
```

主要由 handleMovement()這個函式掌管

如果是 monster room 的話只能選stay or retreat, 所以另外處理 = 直接進入 handle event 內，動用 moster 的 triggerevent (前面的東西~ 只能選 attack or retreat 的攻擊迴圈 & 打怪提示語)

不是 monster room 的話，讓player選擇要前進的方向

"檢查"該方向有沒有房間 (才不會 crash 程式!

有 -> 調用 player 的 changeRoom()

沒有 -> 輸出提示語並返回

```
if( di == 'A' )
{
    if( (player.getCurrentRoom()-> getUpRoom() == NULL) )  // obj
    {
        cout<<"there is no room!";
    }
    else  //there is room
    {
        player.changeRoom(player.getCurrentRoom()-> getUpRoom());
    }
}
```

player 的 changeRoom()

```
void Player::changeRoom(Room* newRoom)
{
    this->previousRoom = this->currentRoom;
    this->currentRoom = newRoom;
    cout<<"Now you are in Room"<<(this->getCurrentRoom()->getIndex())<<endl;
};
```

內部 pointer 的交換與賦予新值 (傳進 function 的是要去的房間) 並輸出現在房間
-> 完成 Movement !

4. Game Logic

```
int main(){
    Dungeon dungeon = Dungeon();
    dungeon.runDungeon();   // run 了以後再create 東西好了
    return 0;
}
```

main 裡只要創一個 Dungeon 的 object 然後 call runDungeon() 就好

```
void Dungeon::runDungeon()
{
    //PlaySound(TEXT("C:\\Users\\Yoona\\Desktop\\Happy 2\\d

    PlaySound(TEXT("HP.wav"), NULL, SND_ASYNC | SND_LOOP);
    startGame();

    while(1)  //while true
    {
        chooseAction();  //deal with the interaction too
        //cout<<"\n\n";
        if( checkGameLogic() )  //while check game logic is
        {
            //mciSendString(TEXT("close HP"), NULL, 0, NULL
            break;
        }
    }
};
```

runDungeon 內先 call startGame()建好 map, player 再進入後面的 while 的 choose action 邏輯迴圈

startGame() 裡面不論如何

先 Creat map 再 Creat Player

如果選擇要 continue,

再將資料改寫就好

**player 的 room 資訊記得在 creat map 內設好~~

```
void Dungeon::runDungeon()
{

    while(1)  //while true
    {
        chooseAction();  //deal
        //cout<<"\n\n";
        if( checkGameLogic() )
        {
            break;
        }
    }
};
```

在 run dungeon 內只要 choose action 一次 (move, interact, check status, save)

就會 check 一次 game logic

如果反回值是 true，讓遊戲結束

game logic 內的判斷

- 與遊戲結束直接相關
1. player 有沒有活著
2. 所有的 monster 都死了且在出口房 -> victory

- 沒直接相關
3. Monster 在戰鬥後死亡，需清理房間 (承 moster triggerevent 中返回主函式的部分，這裡才清房間~)

[1]　if player 死了，回傳 true

```
bool Dungeon::checkGameLogic()  //**目前只check遊戲
{
    if( player.getCurrentHealth() <= 0 ) //player
    {
        cout<<"Game Over, you lose"<<endl;
        return true;
    }
```

[2] if monster 死了，要從房間清掉

```
else if( checkMonsterRoom() ) //monster dead -> pop the monster, and if tl
{
    vector<Object*> obj = (player.getCurrentRoom()->getObjects());
    int i = 0;
    for( auto x : obj )
    {
        if( x->getTag() == "Monster" )
        {
            Monster *m = dynamic_cast<Monster*>(x);  //x 沒有 getCurrentH
            if( m->getCurrentHealth() <= 0 )
            {
                player.getCurrentRoom()->eraseObject(i);  //pop the monst
                i--;
                cout<<"\nYou have defeated the monster in this room!\n";
                player.triggerEvent(&player);
                break;
            }
        }
        i++;
    }
}
```

[3] 判斷是否 victory

```
if( ( !checkMonsterRoom() ) && (player.getCurrentRoom()->getIsExit() == true) )
{
    cout<<"\nYou have finished all game!\nVictory!!\n"<<endl;
    return true;
}  //the last room is monster room

cout<<"\n\n";

return false;  //other situation: return false, the game not yet over
```

清完房間 [2.]做完，才能下來!!

5. Record system

Save

為了方便管理，分成 save player & save Room 兩個部分管理

```
void Record::savePlayer(Player* p, ofstream& o)
{
    //player: name/tag/maxh/currenth/attack/defens
    o<<p->getName()<<" "
    <<p->getTag()<<" "
    <<p->getMaxHealth()<<" "
    <<p->getCurrentHealth()<<" "
    <<p->getAttack()<<" "
    <<p->getDefense()<<" "
    <<p->getCurrentRoom()->getIndex()<<" "
    <<p->getPreviousRoom()->getIndex()<<" "
    <<p->getInventory().size()<<" ";//將資料輸入至

    vector <Item> inventory = p->getInventory();

    for( auto x : inventory )
    {
        //item array: name/tag//health/attack/defe
        o<<x.getName()<<" "
        <<x.getTag()<<" "
        <<x.getHealth()<<" "
        <<x.getAttack()<<" "
        <<x.getDefense()<<" ";
    }
}
```

Save Player

就把需要存的資料按照順序輸出

其中 inventory 的處理則是先輸出現在身上有多少 items (方便 load 回來的 for 迴圈知道要跑幾次)，再依序輸出 items 的特性

```
void Record::saveRooms(vector<Room>& r, ofstream& o)
{
    // cout hm rooms first
    o<<r.size()<<" ";
    for( auto x : r ) //run the room vevtor
    {
        //index/ isExist/ unroom/ downroom/ leftroom/ rightroom / hm object + /n
        o<<x.getIndex()<<" "
        <<x.getIsExit()<<" ";

        if( x.getUpRoom() == NULL ) o<<-1<<" "; else  o<<x.getUpRoom()->getIndex()<<" ";  //**
        if( x.getDownRoom() == NULL ) o<<-1<<" "; else  o<<x.getDownRoom()->getIndex()<<" ";
        if( x.getLeftRoom() == NULL ) o<<-1<<" "; else  o<<x.getLeftRoom()->getIndex()<<" ";
        if( x.getRightRoom() == NULL ) o<<-1<<" "; else  o<<x.getRightRoom()->getIndex()<<" ";
```

save rooms 的部分比較麻煩，先從房間鏈結說起

總之先跑過 room 的 vector 去分別細看每個 room 內要存的東西

1. 房間鏈結 - 遊戲重開後存東西的記憶體位置就會不同了，所以不用存房間記憶體位置這個不實際的方式，也容易 crash 程式

   所以採用的方式是去記上下左右房間的 index 是多少，等 load 回來再把房間們連起來~

```
o<<x.getObjects().size()<<" "; //> 其實應該只會有
vector<Object*> obj = x.getObjects();

for( auto yy : x.getObjects() )
{
    //need to down cast first
    if( yy->getTag() == "Item" )
    {
        Item *y = dynamic_cast<Item*>(yy);
        //tag/ name/ h/ a/ d
        o<<y->getTag()<<" "
        <<y->getName()<<" "
        <<y->getHealth()<<" "
        <<y->getAttack()<<" "
        <<y->getDefense()<<" ";
    }
    else if( yy->getTag() == "Monster" )
    {
        Monster *y = dynamic_cast<Monster*>(yy);
        //tag/ name/ mh/ ch/ a/ d
        o<<y->getTag()<<" "
        <<y->getName()<<" "
        <<y->getMaxHealth()<<" "
        <<y->getCurrentHealth()<<" "
        <<y->getAttack()<<" "
        <<y->getDefense()<<" ";
    }
```

2. 存完房間位置後就是存房間內具體有什麼東東了 -

一樣先輸出有多少東西，方便 load 的 for 迴圈運行

接著遍歷 Object* 的 vector 依照是什麼東西去輸出對應要輸出存檔的東西

而輸出各自要記得的東西前，由於大家要存檔的東西不同，所以先輸出各自的 tag 方便 load 回來時的運作

Item & Monster 就輸出基本屬性就 ok 了

而進行調用屬性操作前要 downcast 一次到 derived class 的層級才能調用各自在 derived class 才有的屬性

其中 NPC 的 save 比較麻煩因為 NPC 自己也有個 commodity vector 要存

不過實現方法一樣，先輸出 vector 多大，在繼續依序輸出每個 item 要儲存的東西

```
void Record::saveToFile(Player* p, vector<Room>& r)
{
    ofstream fp;  //**fstream
    fp.open("record_player.txt", ios::out );  //open
    //**will open a txt in dungeon file
    savePlayer(p, fp);
    fp.close();

    ofstream fr;  //**fstream
    fr.open("record_room.txt", ios::out );
    saveRooms(r, fr);
    fr.close();
}
```

```
void Record::savePlayer(Player* p, ofstream& o)
void Record::saveRooms(vector<Room>& r, ofstream& o)
```

```
else if( yy->getTag() == "NPC" ) //NPC ha
{
    NPC *y = dynamic_cast<NPC*>(yy);
    //tag/ name/ mh/ ch/ a/ d/ commodity
    o<<yy->getTag()<<" ";

    o<<y->getCommodity().size()<<" "; //
    //<<y->getScript()<<" " 先不get scrip
    //npc 也不用 h, a, d ...'

    vector<Item> c = y->getCommodity();
    for( auto z : c ) // all z are items
    {
        //item array: name/tag//health/at
        o<<z.getName()<<" "
        <<z.getTag()<<" "
        <<z.getHealth()<<" "
        <<z.getAttack()<<" "
        <<z.getDefense()<<" ";
    }
}
// +/n
o<<" ";
```

歐忘了說，上述的輸出資料流是 saveToFile 先開好的與 .txt 檔的 ofstream 連結，在傳進 function內使用 (function 內是 o)，使用方式同 cout （ifstream 的部分也差不多操作）

另外存檔的每筆資料的間格是 "空格"，沒有其他符號，方便 load 的處理

Load

這邊的設計是，不論如何遊戲開始時都先 create 一次 player & map, 如果玩家選擇 continue，再將既有資料改寫就好了 (所以很多現成的東西可以直接調用 ex: rooms[?] 的記憶體位置 之類的東西~)

```
void Record::loadFromFile(Player* p, vector<Room>& r)
{
    ifstream fp;
    fp.open("record_player.txt", ios::in );
    loadPlayer(p, r, fp);
    fp.close();

    ifstream fr;
    fr.open("record_room.txt", ios::in );
    loadRooms(r, fr);
    fr.close();
}
```

一樣先開好 .txt 的連結準備讀檔，再將資料流與參數(room, player)傳入進行讀檔及改值的動作

可以改值因為傳的是 address 跟 reference

```
void Record::loadPlayer(Player* p, vector<Room>& r, ifstream& i)
{
    //player: name/tag/maxh/currenth/attack/defense/currentroomindex/r

    string n, t; //name, tag
    int mh, ch, a, d, cr, pr, innum;
    i>>n>>t>>mh>>ch>>a>>d>>cr>>pr>>innum;
    p->setName(n);
    p->setTag(t);
    p->setMaxHealth(mh);
    p->setCurrentHealth(ch);
    p->setAttack(a);
    p->setDefense(d);
    //room in player 不會有NULL的情況 (index == -1 的 situation)
    p->setCurrentRoom(&r[cr]); // room needed to be load first
    p->setPreviousRoom(&r[pr]);
    //其實可以寫 player = Player(.....) 意思同copy, 非初始化那步也可以

    for( int j = 0; j < innum; j++ )
    {
        //item array: name/tag/health/attack/defense
        string nn, tt;
        int hh, aa, dd;
        i>>nn>>tt>>hh>>aa>>dd;
        p->reloadItem( Item(nn, hh, aa, dd) ); //不用tag
        //player 的背包裏面一開始不會有任何東西，所以可以直接加載
    }
}
```

把資料按照順序讀入

*因為 i 的使用方式同 cin , 又每筆資料都有用 "空格" 區分好了, 所以就直接一直讀資料就好~

讀完資料後調用 set function 更新 player 資訊 (currenet/previous room 可以直接給房間位置了

items' inventory 的話就是照順序來 load 回來, 並調用 player 的 reload function

```
void Record::loadRooms(vector<Room>& r, ifstream& i)
{
    // push hm first
    // cuz start game will create all the things first, so only need to renew the information
    int hm;
    i>>hm;

    for( int j = 0; j < hm; j++ )
    {
        //index/ isExist/ uproom/ downroom/ leftroom/ rightroom / hm object + /n
        int index, isExit, uproom, downroom, leftroom, rightroom, hmo;
        i>>index>>isExit>>uproom>>downroom>>leftroom>>rightroom>>hmo;

        r[index].setIndex(index);
        r[index].setIsExit(isExit);
        if( uproom == -1 ) r[index].setUpRoom(NULL); else r[index].setUpRoom(&r[uproom]);
        if( downroom == -1 ) r[index].setDownRoom(NULL); else r[index].setDownRoom(&r[downroom]);
        if( leftroom == -1 ) r[index].setLeftRoom(NULL); else r[index].setLeftRoom(&r[leftroom]);
        if( rightroom == -1 ) r[index].setRightRoom(NULL); else r[index].setRightRoom(&r[rightroom]);
```

load rooms 的載入則是每個房間先利用 index 連結起四周的房間(被 create room 先建好了，所以有 adress 可以用)~~

```
vector <Object*> vec; // new :
if( hmo == 0 )
{
    r[index].setObjects(vec);
    continue; //than return
}
```

沒東西的話直接 set

```
// 每個 room 裡面都只會有一個
// tag first
string tag;
i>>tag; //get the tag first
Object* p;
```

重設房間內 Object 的部分則是令出一個新的 vector，根據房間內 有沒有東西 / 有的話有什麼東西 (tag 協助判斷)，把那新的 vector 設好後，調用 Room 的 setObject 函式完成 room 內 Object* vector 的重新設定

```
void Room::setObjects(vector<Object*> objects)
{
    this->objects = objects;
};
```

```
if( tag == "Item" )
{
    //name/ h/ a/ d
    string name;
    int h, a, d;
    i>>name>>h>>a>>d;
    p = new Item( name, h, a, d );
    vec.push_back(p);
    r[index].setObjects(vec);  // c
}
```

```
else if( tag == "Monster" )
{
    // name/ mh/ ch/ a/ d
    string name;
    int mh, ch, a, d;
    i>>name>>mh>>ch>>a>>d;
    p = new Monster( name, mh, a, d );
    Monster *m = dynamic_cast<Monster*> (p);
    m->setCurrentHealth(ch); // need to use th
    vec.push_back(m);
    r[index].setObjects(vec);
}
```

Monster 跟 Item 的話就直接依據紀錄中的屬性 new 出來給一個 pointer, 把 pointer 丟進 vector 內，再把 vector 丟進 set_function內就完成了

另外 NPC 的 script 因為有換行字元不好讀檔所以就直接不存了~交給 create map 的時候初始化 (寫死了

```
else if( tag == "NPC" )
{
    // NPC 的話只需要改 r 裡面的 item vector
    vector<Item> com;
    int hmo;
    i>>hmo;
    for( int k = 0; k < hmo; k++ )
    {
        //item array: name/tag/health/attack/defense
        string name, tag;
        int h, a, d;
        i>>name>>tag>>h>>a>>d;
        com.push_back( Item(name, h, a, d) );
    }
    NPC *n = dynamic_cast<NPC*> ((r[index].getObjects())[0]);
    n->setCommodity(com);
}
```

NPC 的話因為沒參與戰鬥所以基本屬性都沒被更改，只需要改動 commodity vector 內的東西就好 (被拿走過的東西不能再出現)

所以就是建一個新的 Item vector, 把之前存的 Item 讀回來並用 constructor 直接塞進vector 內

Item vector 設好後，調用 NPC 的 setCommodity 函式完成 Commodity list 的更新

```
void NPC::setCommodity(vector<Item> commodity)
{
    this->commodity = commodity;
}
```

6. 加分 - 音樂

標頭檔

```
#include<Windows.h>
#include<mmsystem.h>
#include<dsound.h>
#pragma comment(lib,"winmm.lib")
```

音樂插入的方式是用 c++ 內可以 include 的標頭檔的裡面的功能~~

但其中 dsound.h 這個標頭檔 code blocks 沒有內建，網路上也找不到可以下載的 .h 包，所以我把作業環境搬到 visual studio 了~

先在 .h 檔內加入這 4 行~

(因為沒有要用到太高級的功能也沒有要視覺化，有些音樂的處理也可以事先編輯好 ex: 淡入淡出 etc.，所以為了省下 intall 新的程式包並設好環境的精力，故選擇此較方便的方式

如果有以上提到的需求，那我應該會選擇使用 SFML)

.cpp 檔

我使用插入音樂的方式有兩種:

1. Playsound
2. mciSendString

1. Playsound
   用於遊戲全程的 **背景音樂**
   只支援 .wav 檔
   在 start game 前開始

由於 PlaySound 自己

1. 不能多線程撥放 (不能同時呼叫多個背景音)
2. 但只要指令下了就會持續撥放音樂, 與程式的運行無關

-> 故選此作背景音

使用方法: (如圖)

其中 SND_LOOP 是循環撥放的意思
要結束音樂的話就把檔名那邊設成NULL 就 ok 了
(ex: victory 音效前將其設成 NULL

```
PlaySound(TEXT("HP.wav"), NULL, SND_ASYNC | SND_LOOP);
startGame();
```

```
cout<<"\nYou have finished all game!\nVictory!!\n"<<endl;
PlaySound(NULL, NULL, SND_FILENAME);
```

```
mciSendString(TEXT("open final.mp3 alias final"), NULL, 0, NULL);
mciSendString(TEXT("play final repeat"), NULL, 0, NULL);
Sleep(20000);
mciSendString(TEXT("close final"), NULL, 0, NULL);
```

2. mciSendString

必須加上 Sleep 才能聽到音樂
->與**程式運行相關了**，其他程式必須等音樂播完才能繼續執行

**但聲音可以與 playsound 的背景音一起出現不會互相干擾

-> 用於**音效音**

使用方法 : (每次呼叫共四行，支持.mp3 檔)

1 : 打開 XXX.mp3 別名是 OO (之後音樂在程式中叫OO)
2 : play OO (repeat 可以重複)
3 : Sleep(1000) 程式停止 1000毫秒(1 秒) 聽音樂撥放
4: 關閉 OO

我的音效音:
換房間的腳步聲，怪物出現的吼叫聲，打鬥聲，開寶箱的聲音，NPC 的笑聲，lose / victory 時的音效

新知識

我一開始加音檔的時候怎麼加怎麼不成功，但至少會一直播報錯音，上網查到的資料不多，但可以試的方式都試了，差點快放棄。

最後是發現程式檔及音樂檔都必須要存在 visual studio 的預設儲存路徑：

C/User/Yoona/source/repos 下，程式才找的到位置

不然存在其他地方即使給了絕對位置好像還是找不到音檔~

result

額不知道前面寫的夠不夠多，總之程式不是很順利的跑起來了就是在我
精密的 Debug 技術下 (= cout 大法 + 我聰明的腦袋) 跑起來了~~
詳細呈現在影片中可觀賞悠~

Disscussion – 淺提遇到的小問題/思考

1. Layer3 的 constructor 要怎麼對 Layer1 的 private member 賦值

   ans : L3 先丟給 L2，再由 L2 丟給 L1 (L2 可在 member initialize list call L1's constructor, L3 不行

   dis : 那時候就每種方法都試試，就試出來原來成員初始化列表那邊原來一次只能向上 call 一層的 constructor~

2. 鏈結主程式

   我記得那時候寫完各個 class 的 function 後，對我來說最困難的地方是把程式連結起來，有些東西很不知道要寫在哪個 function 內(ex: triggerevent 的腳色提示語)才不會一直重複輸出看起來很怪，還有程式主環節的邏輯要怎麼串，返回是回到程式的哪裡之類的東西，很怕自己寫錯要改很久於是頭很痛...

最後是室友跟我說 : 反正只要能好好實現功能就是好程式阿~ 所以我才想說那還想不清楚就算了, 總之先寫下去就對了~

最後就是很多東西就真的是等呈現出來才知道哪裡的邏輯卡卡的再去慢慢修 (我覺得這好像真的是我的弱點XD, 修的特別久, 我記得有花了一天哈哈哈), 所以一定要先寫出來才能知道錯哪阿~~(這也是我寫這份功課的小習慣來源 : 不到週末不寫XD, 因為怕邏輯斷斷續續的要修的東西會變更多, 而且我如果沒有長時間沉浸在裡面, 邏輯會接不上, 一定會寫的超卡而且 bug 滿天飛...XD)

另外寫 report 的時候又有重新梳理了一次流程, 所以對自己寫出來的東西又更清楚的瞭解了一次運作原理, 我覺得對之後如果還要設計一次遊戲, 這個步驟很重要!(重新梳理邏輯 + 附上所有對 code 的說明) 也算是變相的給自己留下了翻閱紀錄, 以後還可以回來看當初的自己是怎實現這個遊戲的!~

3. Room Record

我第三個遇到的大問題是 Room Record 哈哈，那時候 Player 的都可以好好存檔，但 Room 的 .txt 檔怎麼看怎麼空白XD

最後是請出了 cout 大法，一行一行塞，看程式到底是到底是到哪裡 crash 了，最後發現是忘記判如果指向的是 NULL 的話那就存 -1 不能再去要該房間的 index 了~~

4. Room Load

寫 load 前我也是超挫 = =，感覺就會很出事 (而且我所有的資料都是用空白隔開，為了方便讀回來不用額外處理，但人工閱讀.txt 就會很不便)，結果真的出事了XD

兩個 bug 都是室友幫我揪出來的，因為本人已經寫到頭暈XD

1. object 的重設要調用 set function，直接賦值的話因為傳進來的只有 room 是 reference , room 下的 object 不是 reference 不會改到阿~

2. 有一個地方我 continue 寫成 return，自己卡了好久想說為什麼怪物還在XD

結果室友用 1 分鐘就看到那個 return 問我為什麼XD

結論是當你自己寫程式寫到快瘋掉的時候，請求好朋朋的協助也是非常好的方式呢哈哈哈~~

5. 音樂讀不到檔

問題如前面 加分 topic 所述，最後又是室友運用上學期她們班助教說過的 visual studio 存檔注意事項拯救了我 QQ
不然我可能真的要查+修很久了XD (網路上資料其實也不多)

Conclusion

總而言之這次的作業真的蠻好玩的，也讓我紮紮實實的體驗到了 物件導向 的編程架構及精神，真的真的學到很多東西，而這個經驗也超珍貴、必備及實用。非常謝謝助教出這個功課給我們寫，真的很開心有自己寫出一個小遊戲的酷經驗了呢~~

另外 coding 過程中難免遇到很多問題，讓我知道"耐心" 是一個非常重要的東西，而我還需要繼續努力。還有不要不好意思開口向別人詢問，很多時候同學就剛好會有我們所欠缺的知識 or 因為不是自己卡住的東西，所以能夠快速地找出哪裡有錯~ 所以同儕之間的互相幫助也是非常重要的呢! 在此感謝我的朋朋+室友~~ 也覺得當自己有能力時，一定要多多分享or 幫忙別人~~

最後就是在跟朋友互相玩遊戲的時候，就會發現大家都好優秀XD，有些人邏輯介面承接的很好，有些人程式內部設計簡潔漂亮，有些人自學了視覺化還做的超厲害，有些人遊戲故事精彩有趣有創意...，我看到大家身上好多可以學習的地方，也迫不期待在等比較閒的時候，可能問問看能不能借他們的code 來看，學習別人的優點，然後變得更厲害~~ :D