

# Dungeon Report

112550109 楊榮竣

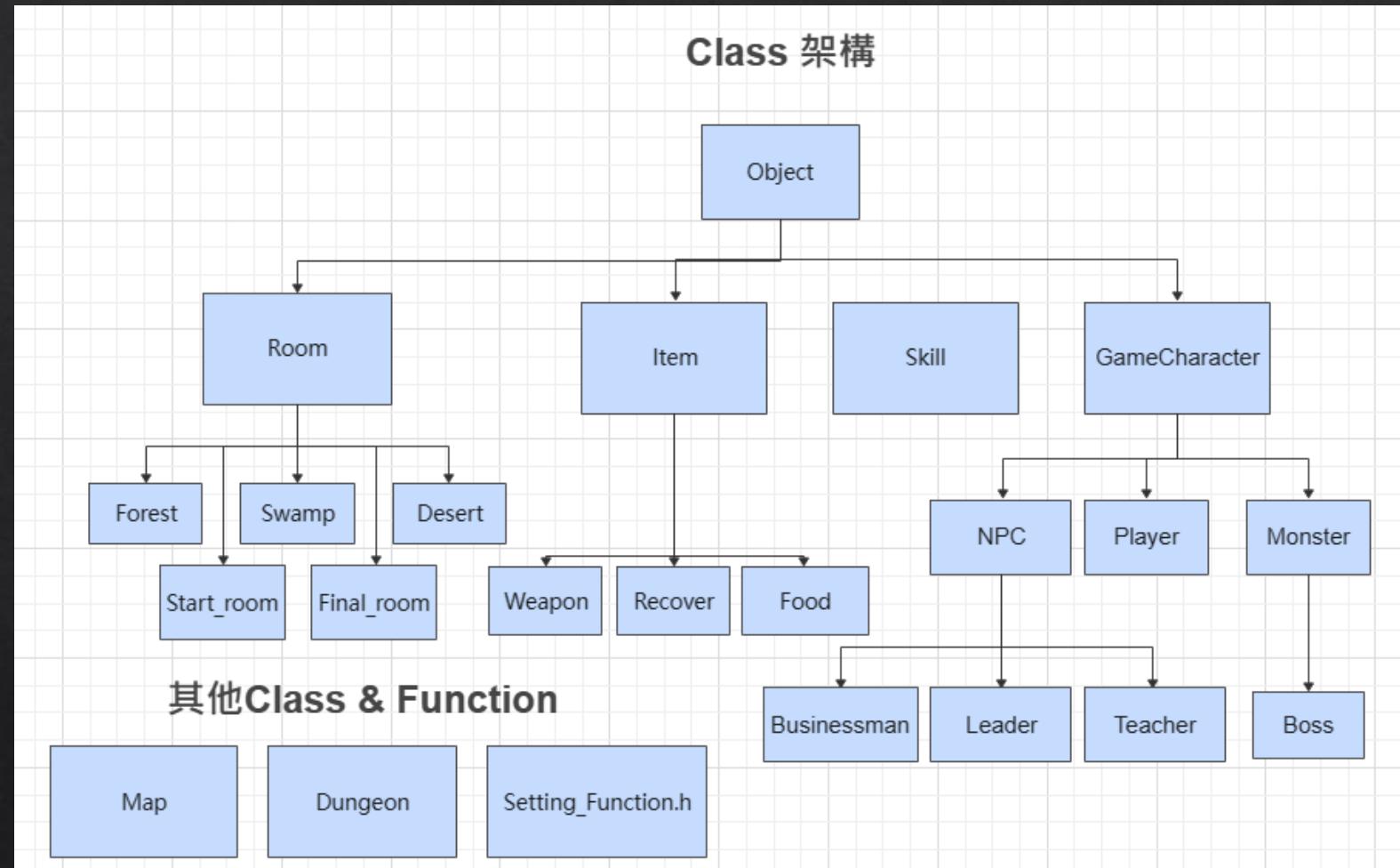
# Outline

1. Class 的結構與成員
2. Basic Function
3. Optional Enhance
4. Discussion
5. Conclusion

# Class的結構與成員

以 Object 為最底層的結構，其他 class 繼承  
Object 在各自增加新功能函式，組合出  
Dungeon 遊戲

```
class Object {  
public:  
    virtual bool trigger_event(Object*) = 0;  
    Object();  
    Object(string , string);  
    //setter  
    void set_name(string );  
    void set_tag(string );  
    //getter  
    string get_name();  
    string get_tag();  
private:  
    string name;  
    string tag;  
};
```



# Class的結構與成員

Setting\_Funvtion.h :

與數值相關的設定，都放在這些函式中，方面調整與設定每一項數值。只要輸入type就可以得到對應的物件。

```
//Monster setting ( type : 0 ~ 5 )
Monster* Monster_setting(int type) {
    Monster* temp = NULL;
    switch (type) {
        case 0: //野狼
            temp = new Monster("野狼", 20, 10, 5);
            break;
        case 1: //老虎
            temp = new Monster("老虎", 40, 15, 5);
            break;
        case 2: //黑熊
            temp = new Monster("黑熊", 40, 15, 10);
            break;
        case 3: //獅子
            temp = new Monster("獅子", 50, 20, 10);
            break;
        case 4: //蜘蛛
            temp = new Monster("蜘蛛", 30, 10, 5);
            break;
        case 5: //野豬
            temp = new Monster("野豬", 40, 15, 10);
            break;
    default:
        //Debug
        cout << "Type Error(Monster_setting)" << type << endl;
        exit(1);
    }
    if (temp == NULL) {
        cout << "temp error(Monster_setting)" << endl;
    }
    return temp;
}
```

```
// 0~2
Businessman* Businessman_setting(int type);

// 0~2
Teacher* Teacher_setting(int type);

//helping function
void push_back(vector<Skill*>& skills, Skill* skill);
void push_back(vector<Item*>& items, Item* item, vector<int>& costs, int cost);

//Item setting
//Food setting (0~6)
Food* Food_setting(int type);

Weapon* Weapon_setting(int type);

Skill* Skill_setting(int type);

//GameCharaacter setting
Player* Player_setting(int type);

//Monster setting (0~5)
Monster* Monster_setting(int type);

//Room setting
// area 0 : Desert (0~3)
// area 1 : Forest (0~4)
// area 2 : Swamp (0~3)
Room* Room_setting(int area, int type, int index);
```

# Basic Function

- 1. 動作選擇
- 2. 玩家移動
- 3. 顯示狀態
- 4. 檢起與使用物品
- 5. 遊戲地圖生成
- 6. 戰鬥系統
- 7. NPC
- 8. Game Logic
- 9. Hunger System
- 10. Room System Design

# 1. 玩家動作選擇

透過

Room::get\_objects()  
取得房間中所有的物品，並利用迴圈判斷每個物品的 tag，在依據是否偵測到相關物件，顯示可供玩家選擇的選項。

```
//return 1:move to new room
bool Dungeon::choose_action(Player* player, vector<Object*> objects) {
    int choice, input = 0, count=1;
    cout << count++ << ". 顯示角色狀態" << endl;
    cout << count++ << ". 打開背包(使用物品)" << endl;
    cout << count++ << ". 前往下一個房間" << endl;
    cout << count++ << ". 休息(回復活力，會消耗 Hunger 和 Thirst):" << endl;
    for (int i = 0; i < objects.size(); i++) {
        if (objects[i]->get_tag() == "NPC") {
            cout << count++ << ". 與 NPC 對話 ("<< objects[i]->get_name() << ")" << endl;
            break;
        }
    }
    for (int i = 0; i < objects.size(); i++) {
        if (objects[i]->get_tag() == "Weapon" || objects[i]->get_tag() == "Food" || objects[i]->get_tag() == "Recorver") {
            cout << 6 << ". 撿起物品" << endl;
            break;
        }
    }
    cout << "請選擇動作:" << endl;
```

- 1. 顯示角色狀態
  - 2. 打開背包(使用物品)
  - 3. 前往下一個房間
  - 4. 休息(回復活力，會消耗 Hunger 和 Thirst):
  - 5. 與 NPC 對話 (村長)
  - 6. 撿起物品
- 請選擇動作：

- 1. 顯示角色狀態
  - 2. 打開背包(使用物品)
  - 3. 前往下一個房間
  - 4. 休息(回復活力，會消耗 Hunger 和 Thirst):
  - 5. 與 NPC 對話 (村長)
- 請選擇動作：

- 1. 顯示角色狀態
  - 2. 打開背包(使用物品)
  - 3. 前往下一個房間
  - 4. 休息(回復活力，會消耗 Hunger 和 Thirst):
- 請選擇動作：

# 1. 玩家動作選擇

玩家輸入選擇，並透過 switch 及各個與功能相關的程式。執行完選擇的選項。

1. 顯示狀態 player->show\_status()
2. 打開背包(使用物品) player->show\_bag()
3. 腳色移動 handle\_movement(player)
4. 休息 player->take\_break()
5. 與NPC互動 objects[0]->trigger\_event(player)
6. 撿起物品 take\_up\_object(player)

```
cin >> choice;
switch (choice) {
case 1:
    player->show_status();
    break;
case 2:
    player->show_bag();
    int choice;
    while (1) {
        if (player->get_inventory().size() == 0) { break; }
        cout << "請輸入想使用的物品(輸入-1返回):" << endl;
        cin >> choice;
        if (choice == -1) { break; }
        if (choice<0 || choice >player->get_inventory().size()) {
            cout << "輸入錯誤，請重新輸入。" << endl;
        }
        else {
            player->use_item(choice-1);
            break;
        }
    }
    break;
case 3: //角色移動
    if (player->get_cur_power() == 0) {
        cout << "活力不足，無法移動" << endl;
    }
    else {
        cout << endl;
        handle_movement(player);
        return 1;
    }
    break;
case 4:
    player->take_break();
    break;
case 5:
    //Debug
    if (objects[0]->get_tag() != "NPC") {
        cout << "object tag error " << endl;
        exit(1);
    }
    //與NPC互動
    objects[0]->trigger_event(player);
    break;
case 6:
    take_up_object(player);
    break;
}
```

## 2. 玩家移動

利用

Room :: test\_path()

偵測房間是否有通道，並將所有的相鄰的房間放進 rooms 中，輸出選項，供玩家選擇。

Test\_path(int n)  
依據輸入的數字不同，決定偵測方向，並回傳是否有通道。

```
bool Room::test_path(int n) {
    //0:up 1:down 2:right 3:left
    switch (n) {
        case 0:
            if (up_room) { return 1; }
            else { return 0; }
            break;
        case 1:
            if (down_room) { return 1; }
            else { return 0; }
            break;
        case 2:
            if (right_room) { return 1; }
            else { return 0; }
            break;
        case 3:
            if (left_room) { return 1; }
            else { return 0; }
            break;
        default:
            return 0;
    }
}
```

```
void Dungeon::handle_movement(Player* player) {
    player->show_map();
    for (int i = 0; i < 4; i++) {
        if (player->get_current_room()->test_path(i)) {
            switch (i) {
                case 0:
                    cout << choice << ". Go up." << endl;
                    rooms.push_back(player->get_current_room()->get_up_room());
                    break;
                case 1:
                    cout << choice << ". Go down." << endl;
                    rooms.push_back(player->get_current_room()->get_down_room());
                    break;
                case 2:
                    cout << choice << ". Go right." << endl;
                    rooms.push_back(player->get_current_room()->get_right_room());
                    break;
                case 3:
                    cout << choice << ". Go left." << endl;
                    rooms.push_back(player->get_current_room()->get_left_room());
                    break;
            }
            choice++;
        }
    }
    cout << "請選擇移動方向(輸入-1返回): ";
}
```

## 2. 玩家移動

透過 Room 的成員函式找到可行走的路徑，並儲存在rooms中，再將路線顯示。隨後偵測玩家輸入，在利用 change\_room(Room\*) 達到移動的功能。

```
void Dungeon::handle_movement(Player* player) {
    //玩家輸入選項
    while (1) {
        cin >> your_choice;
        if (your_choice <= rooms.size() && your_choice > 0) { break; }
        else if (your_choice == -1) { break; }
        cout << "輸入錯誤，請從新輸入: ";
    }

    //player 移動
    if (your_choice != -1) {
        player->change_room(rooms[your_choice - 1]);
        cout << "移動到新房間" << endl;
    }
}

void Player::change_room(Room* room) {
    previous_room = current_room;
    current_room = room;
    current_room->set_go_throough(2);
    if (previous_room != NULL) { previous_room->set_go_throough(1); }
    map->update_map(current_room, previous_room);
}
```

### 3. 顯示狀態

```
void Player::show_status()
```

Player 類別的成員函式，負責顯示角色的各種狀態狀態，例如：名字、職業、血量等。此外，還有顯示背包內的物品，以及特殊狀態包含虛弱、飢餓、中毒、無力。

```
//bag  
show_bag();
```

↑ 顯示背包內物品

依照poison的數  
值顯示不同文字

```
cout << "Poison(中毒狀態): " << poison;  
switch (poison) {  
case 0:  
    cout << " (未中毒)" << endl;  
    break;  
case 1:  
    cout << " (植物中毒)" << endl;  
    break;  
case 2:  
    cout << " (怪物中毒)" << endl;  
    break;  
}
```



```
cout << endl << "角色狀態:" << endl;  
if(cur_hunger<max_hunger/2){  
    cout << count++ << ". 飢餓(自然回復MP減半)" << endl  
}  
  
if (cur_thirst <= 3) {  
    cout << count++ << ". 虛弱(無法使用技能)" << endl;  
}  
if (poison != 0) {  
    cout << count++ << ". 中毒(隨著移動損血)" << endl;  
}  
if (cur_power < max_power / 2) {  
    cout << count++ << ". 疲勞(攻擊力減半)" << endl;  
}  
if (cur_power == 0) {  
    cout << count++ << ". 無力(無法移動)" << endl;  
}  
//  
cout << endl << "-----" << endl;
```

狀態說明：

飢餓：Hunger 少於最大值一半

虛弱：Thirst <= 3

中毒：poison != 0

疲勞：power 少於一半

無力：power 等於0

### 3. 顯示狀態

```
void Player::show_bag()
```

Player 的成員函式，  
運用for迴圈得到  
inventory 內的物品，  
並偵測物品的 tag  
依照 tag 的不同顯  
示不同的文字。

```
void Player::show_bag() {
    int count;
    cout << "Item:" << endl;
    if (inventory.size() == 0) {
        cout << "No Item." << endl;
    }
    else {
        count = 1;
        for (int i = 0; i < inventory.size(); i++) {
            if (inventory[i]->get_tag() == "Weapon") {
                if (inventory[i]->get_value() == 1) {
                    cout << i + 1 << ". " << "武器 " << inventory[i]->get_name() << "\t\t已裝備"
                }
                else {
                    cout << i + 1 << ". " << "武器 " << inventory[i]->get_name() << "\t\t未裝備"
                }
            }
            else if(inventory[i]->get_tag() == "Recover") {
                cout << i + 1 << ". " << "藥水 " << inventory[i]->get_name() << "\t回復數值: "
            }
            else if (inventory[i]->get_tag() == "Food") {
                cout << i + 1 << ". " << "食物 " << inventory[i]->get_name() << "\t\t回復數值: "
            }
        }
    }
}
```

# 4. 檢起物品

## Dubgeon::take\_up\_object(Player\*)

```
//處理檢起物品功能
bool Dungeon::take_up_object(Player* player) {
    int choice,count,num;
    Item* temp=NULL;
    if (player->get_current_room()->get_objects().size() == 0) { cout << "房間內空無一物" << endl; return 1; }
    while (1) {
        num=player->get_current_room()->show_objects();
        cout << "請選擇要檢起的物品(輸入-1返回):" << endl;
        cin >> choice;
        if (choice == -1) { break; }
        else if (choice <= 0 || choice > num) {
            cout << "輸入錯誤，請重新輸入。" << endl;
        }
        else {
            for (int i = 0; i < player->get_current_room()->get_objects().size(); i++) {
                if (player->get_current_room()->get_objects()[i]->get_tag() == "NPC" || player->get_current_room()->get_objects()[i]->get_tag() == "Monster") //將tag不相符的物件過濾掉
                    continue;
                choice--;
                if (choice == 0) {
                    temp = dynamic_cast<Item*>(player->get_current_room()->get_objects()[i]);
                    player->get_current_room()->get_objects().erase(player->get_current_room()->get_objects().begin() + i);
                    break;
                }
            }
            player->add_item_normal(temp);
            cout << "成功檢起物品" << endl;
            break;
        }
    }
    return 1;
}
```

將tag不相符的物件過濾掉

偵測房間內是否有 NPC 或 Monster 以外的 object 並顯示在 cmd 上，接著依據玩家的輸入找出物品，利用 add\_item\_normal() 將物品檢起，在刪除原來在房間的物品。

```
void Player::add_item_normal(Item* item) {
    inventory.push_back(item);
}
```

# 4. 使用物品

透過 tag 和 name 來判斷物品是用後會發生的事件。

```
void Player::use_item(int pos) {
    //藥水類
    if (inventory[pos]->get_name() == "HP回復藥水") {
        set_cur_hp(get_cur_hp() + inventory[pos]->get_value());
        cout << get_name() << " 使用 " << inventory[pos]->get_name() << " 回復 " << inventory[pos]->get_value() << " 點 HP" << endl;
        inventory.erase(inventory.begin() + pos);
    }
    else if (inventory[pos]->get_name() == "MP回復藥水") {
        if (get_cur_mp() + inventory[pos]->get_value() > get_max_mp()) {
            set_cur_mp(get_max_mp());
        }
        else {
            set_cur_mp(get_cur_mp() + inventory[pos]->get_value());
        }
        cout << get_name() << " 使用 " << inventory[pos]->get_name() << " 回復 " << inventory[pos]->get_value() << " 點 MP" << endl;
        inventory.erase(inventory.begin() + pos);
    }
    //解毒類
    else if (inventory[pos]->get_name() == "牛奶" || inventory[pos]->get_name() == "解毒草") {
        poison = 0;
        cout << " 使用 " << inventory[pos]->get_name() << endl;
        cout << " 成功解毒" << endl;
        inventory.erase(inventory.begin() + pos);
    }
    //回復 thirst
    else if (inventory[pos]->get_name() == "水") {
        cur_thirst += inventory[pos]->get_value();
        if (cur_thirst > max_thirst) { cur_thirst = max_thirst; }
        inventory.erase(inventory.begin() + pos);
    }
}
```

1. 回復藥水: 依據 name 選擇恢復 HP 或 MP
2. 解毒 : 判斷食物名稱是否為牛奶或解毒草
3. 回復 Thirst : 判斷 name 為水
4. 回復 Thirst & Hunger : 判斷 name 為仙人掌
5. 其他食物 : 偵測 tag , 回復 Hunger
6. 裝備 : 偵測 tag 為武器 , 使用後裝備該武器。若以裝備 , 再次使用會將裝備脫下

```
else if (inventory[pos]->get_name() == "仙人掌") {
    cur_thirst += inventory[pos]->get_value();
    cur_hunger += inventory[pos]->get_value();
    inventory.erase(inventory.begin() + pos);
}
//回復 hunger
else if (inventory[pos]->get_tag() == "Food") {
    cur_hunger += inventory[pos]->get_value();
    if (cur_hunger > max_hunger) { cur_hunger = max_hunger; }
    inventory.erase(inventory.begin() + pos);
}
//武器類
else if (inventory[pos]->get_tag() == "Weapon") {
    Weapon* temp = dynamic_cast<Weapon*>(inventory[pos]);
    if (temp->get_equipment() == 1) {
        temp->set_equipment(0);
        increase_status(-1*temp->get_health(), -1*temp->get_attack(), -1*temp->get_defense());
        cout << " 脫下裝備" << endl;
    }
    else {
        temp->set_equipment(1);
        increase_status(temp->get_health(), temp->get_attack(), temp->get_defense());
        cout << " 穿上裝備" << endl;
    }
}
//避免超過數值上限
if (get_cur_hp() > get_max_hp()) { set_cur_hp(get_max_hp()); }
if (get_cur_thirst() > get_max_thirst()) { set_cur_thirst(get_max_thirst()); }
if (get_cur_hunger() > get_max_hunger()) { set_cur_hunger(get_max_hunger()); }
```

進行數值設定，將超過上限的數值，設定為上限。

# 5. 地圖生成

Room :

負責處理生成房間、加入物品等，每個地區都有需求的功能。

Room\_setting():  
依據輸入的數值不同  
生成不同種類的房間。

```
class Room:public Object{
public:
    //Constructor
    Room();
    Room(string,bool, int, vector<Object*>);
    //virtual function
    virtual bool trigger_event(Object*) override;
    virtual int get_type();
    //檢測房間的通道 (模式: 0:up 1:down 2:right 3:left)
    //return 1:有通道 0:無通道
    bool test_path(int);
    //隨機生成房間
    int randomly_set_room(Room*, Room*, vector<int>);
    //尋找房間關係
    //return ( U:在上方 D:在下方 L:在左方 R:在右方 N:無 )
    char find_relation(Room*);
    //在房間中加入物件
    void add_object(Object* );
    //顯示房間中的所有物件
    int show_objects();
```

```
//getter & setter
int get_go_through();
int get_index();
bool get_is_exit();
bool get_event_happen();
Room* get_up_room();
Room* get_down_room();
Room* get_right_room();
Room* get_left_room();
vector<Object*>& get_objects();
void set_go_throuugh(int);
void set_is_exit(bool);
void set_index(int);
void set_event_happen(bool);
void set_up_room(Room* );
void set_down_room(Room* );
void set_right_room(Room* );
void set_left_room(Room* );
void set_objects(vector<Object*>);
```

```
Room* Room_setting(int area, int type,int index) {
    Room* temp = NULL;
    switch (area) {
        case 0:
            temp = new Desert(type);
            break;
        case 1:
            temp = new Forest(type);
            if(type==3){
                temp->add_object(Monster_setting(4));
            }
            break;
        case 2:
            temp = new Swamp(type);
            break;
        default:
            cout << "area error(Room_setting)" << endl;
            exit(1);
    }
    temp->set_index(index);
    return temp;
}

private:
    int go_through;
    Room* up_room;
    Room* down_room;
    Room* right_room;
    Room* left_room;
    bool is_exit; //出口
    int index; //房間編號
    //Gamecharacter : monster 可能有很多個 NPC 只會有一個
    //Item : 打倒Monster
    vector<Object*> objects;

    //紀錄事件是否發生
    bool event_happen;
```

# 5. 地圖生成

Randomly\_set\_room():

輸入Room\* : 要連接房間的記憶體位置

vector<int>restriction : 在 Map 上已經生成房間的位置

例: 上和左有房間 · { 0 , 2 }

上: 0 下: 1 左: 2 右: 3

隨機一個通道和  
輸入的房間連接 ,  
偵測通道上是否  
為NULL · 且在  
Map 上通道對應  
位置上沒有已經  
生成的房間 。

```
int Room::randomly_set_room(Room* new_room, Room* this_room, vector<int> restriction) {
    vector<int> nums;
    int rand_number = -1;
    bool flag;
    for (int i = 0; i < 4; i++) {
        if (!test_path(i)) {
            nums.push_back(i);
        }
    }
    if (nums.size() != 0) {
        srand(time(NULL));
        while (1) {
            if (restriction.size() == 4) {
                rand_number = -1;
                break;
            }
            flag = 1;
            rand_number = rand() % nums.size();
            for (int i = 0; i < restriction.size(); i++) {
                if (nums[rand_number] == restriction[i]) {
                    flag = 0;
                    break;
                }
            }
        }
    }
}
```

```
    }
    if (flag) { break; }
}
if (rand_number == -1) { return 0; }
switch(nums[rand_number]){
case 0:
    up_room = new_room;
    new_room->set_down_room(this_room);
    break;
case 1:
    down_room = new_room;
    new_room->set_up_room(this_room);
    break;
case 2:
    right_room = new_room;
    new_room->set_left_room(this_room);
    break;
case 3:
    left_room = new_room;
    new_room->set_right_room(this_room);
    break;
}
return 1;
}
else {
    //there is no NULL path.
}
return 0;
}
```

# 5. 地圖生成

將生成的房間儲存在 rooms 中，並用 rand() 生成出隨機數字，代表下一個房間要連接第幾號房間。最後，在用 randomly\_set\_room() 將房間隨機連接至可連接的位置。



```
void Dungeon::create_map(Player* player) {
    srand(time(NULL));
    //variable setting
    int room_count = room_number[mode];
    Room* room;
    vector<Room*> rooms;
    Map setting_map(3,3);
    int create_room_pos;
    Pos pos;
    int type=0;
    int area_number[3] = { 2,3,3 };
    //start room setting
    room=new Start_room();
    player->change_room(room);
    pos = Pos(1, 1);
    player->get_map()->set_map(room, pos);
    rooms.push_back(room);
    setting_map.set_map(room, pos);
    int random_number; //choosing which room is connected.
    int times = 1;
    int i = 0;
    //
    while (times!=room_count) {
        //隨機選擇一個以生成的房間
        random_number = rand() % rooms.size();
        if (times == room_count - 1) {
            //final room setting
            room = new Final_room(room_count - 1);
            //避免 final_room 出現在 start_room 旁
            while (random_number == 0) {
                random_number = rand() % (rooms.size()-1) + 1;
            }
        } else {
            room = Room_setting(i,type,times);
            if (i == 2 && type == area_number[i]) {
                i = rand() % 9;
                if (i == 1 || i == 2 || i == 5) { i = 0; }
                else if (i == 0 || i == 6 || i == 7) { i = 1; }
                else { i = 2; }
                type = rand() % area_number[i];
            }
            else if (type == area_number[i]) {
                type = 0;
                i ++ ;
            }
            else {
                type++;
            }
        }
    }
}
```

# 5. 地圖生成

依序生成商人、師傅、怪獸。  
在迴圈中生成一個商人，  
`b_number` 就減一，等 `b_number` 減至零。就換生成師傅，以此類推。生成出的 GameCharacter 會隨機放入還未被添加 NPC 的房間中。生成出 GameCharacter 會在 `setting_function` 中的類型中隨機選擇。



## 其中一種Businessman

```
case 0:  
    food_item = Food_setting(4);  
    push_back(good, food_item, cost, 30);  
    food_item = Food_setting(3);  
    push_back(good, food_item, cost, 20);  
    food_item = Food_setting(1);  
    push_back(good, food_item, cost, 10);  
    recover_item = new Recover(20, 1);  
    push_back(good, recover_item, cost, 30);  
    recover_item = new Recover(20, 2);  
    push_back(good, recover_item, cost, 40);  
    temp = new Businessman(good, cost);  
    weapon_item = new Weapon("逐暗者", 30, 40, 50);  
    push_back(good, weapon_item, cost, 150);  
    break;
```

```
    room->set_index(times);  
    //cout << random_number << endl;  
    create_room_pos=rooms[random_number]->randomly_set_room(room, rooms[random_number], setting_map.check_near_room(setting_map.search(rooms[random_number])));  
    if (create_room_pos != 0) {  
        setting_map.update_map(room,rooms[random_number]);  
        rooms.push_back(room);  
        times++;  
    }  
}  
cout << "-----" << endl;  
//GameCharacter setting  
int b_number = businessman_number[mode];  
int t_number = teacher_number[mode];  
int m_number = monster_number[mode];  
int number_gamecharacter = b_number + t_number + m_number; //NPC number setting  
GameCharacter* temp;  
//Debug  
if (number_gamecharacter > rooms.size() - 2) { cout << "number_NPC error" << endl; exit(1); }  
//  
for (int i = 0; i < number_gamecharacter, i++) {  
    if (b_number) {  
        random_number = rand() % 3;  
        temp = Businessman_setting(random_number);  
        b_number--;  
    }  
    else if(t_number) {  
        random_number = rand() % 3;  
        temp = Teacher_setting(random_number);  
        t_number--;  
    }  
    else {  
        random_number = rand() % 4;  
        temp = Monster_setting(random_number);  
    }  
    cout << "Random_number : " << random_number << endl;  
    while (1) {  
        random_number = (rand() % (rooms.size() - 2)) + 1;  
        //Debug  
        if (random_number < 0 || random_number >= rooms.size()) {  
            cout << "Random_number : " << random_number << endl;  
            exit(1);  
        }  
        //  
        if (rooms[random_number]->get_objects().size() == 0 || (rooms[random_number]->get_objects().size()==1 && rooms[random_number]->get_objects()[0]->get_name()=="蜘蛛")) {  
            break;  
        }  
    }  
    rooms[random_number]->get_objects().push_back(temp);  
}
```

# 6. 戰鬥系統

戰鬥系統：

由 Monster::trigger\_event() 觸發，採用回合制戰鬥。玩家可以選擇單純攻擊、使用技能、使用物品，作為當前回合的行動。在攻擊過程中，會依照角色狀態，來造成不同傷害量。並用 GameCharacter::cause\_damage() 來到造成傷害的效果。

```
void Monster::fighting_system(Player* player) {
    //variable setting
    int round = 1;
    int choice, choice1;
    int count;
    int rand_number;
    srand(time(NULL));
    //歸零CD時間
    for (int i = 0; i < player->get_skills().size(); i++) {
        player->get_skills()[i]->set_cur_CD(0);
    }
    //戰鬥開始
    cout << "進入戰鬥" << endl;
    while (1) {
        //顯示戰鬥基礎資訊(Player status and Monster status)
        player->show_status();
        show_status();
        cout << "Round " << round << " :" << endl;
        //Player choose action
        choice = 0;
```

```
while (choice <= 0 || choice > 3) {
    cout << "1. 攻擊" << endl;
    cout << "2. 使用技能" << endl;
    cout << "3. 使用道具" << endl;
    cout << "請選擇這回合的動作：" ;
    cin >> choice;
    switch (choice) {
        case 1: //攻擊
            cout << player->get_name() << " 發動攻擊" << endl;
            if (player->get_cur_power() < player->get_max_power() / 2) {
                cause_damage(player->get_atk() * 2 / 3); // 活力值少於 1/2 攻擊力會下降 1/3
            } else {
                cause_damage(player->get_atk()); //活力值正常
            }
            break;
        case 2: //使用技能
            player->show_skill();
            while (1) {
                //偵測是否為虛弱狀態
                if (player->get_cur_thirst() <= 3) { cout << "虛弱狀態中，無法使用技能" << endl; choice = 0; break; }
                //玩家輸入
                cout << "請選擇想使用的技能(輸入-1返回)：" ;
                cin >> choice1;
                //處理輸入-1離開的功能
                if (choice1 == -1) {
                    choice = 0;
                    break;
                }
                //檢測玩家輸入為有效數值
                else if (choice1 < 1 || choice1 > player->get_skills().size()) {
                    cout << "輸入錯誤，請重新輸入" << endl;
                }
                //CD 時間限制玩家使用技能
                else if (player->get_skills()[choice1 - 1]->get_cur_CD() != 0) {
                    cout << player->get_skills()[choice1 - 1]->get_name() << " : 技能CD中，無法使用" << endl << endl;
                }
                //檢測MP充足
                else if (player->get_cur_mp() < player->get_skills()[choice1 - 1]->get_cost()) {
                    cout << "MP不足，無法使用技能" << endl;
                }
            }
        }
    }
}
```

偵測到虛弱狀態，離開迴圈

# 6. 戰鬥系統

活力數值偵測，少於1/2傷害減少1/3

```
else {
    if ((player->get_cur_power() < player->get_max_power() / 2) {
        cause_damage(player->use_skill(choice1 - 1) * 2 / 3); // 活力值少於 1/2 攻擊力
    }
    else {
        cause_damage(player->use_skill(choice1 - 1)); //活力值正常
    }
    break;
}
break;
case 3: //使用道具 只顯示藥水類物品
count = player->show_bag_fight();
while (1) {
    cout << "請選擇想使用的道具(輸入-1返回): ";
    cin >> choice1;
    if (choice1 == -1) {
        choice = 0;
        break;
    }
    else {
        if (choice1 < 1 || choice1 > count) {
            cout << "輸入錯誤，請重新輸入" << endl;
        }
        else {
            for (int i = 0; i < player->get_inventory().size(); i++) {
                if (player->get_inventory()[i]->get_tag() == "Recover") {
                    choice1--;
                }
                if (choice1 == 0) {
                    player->use_item(i);
                }
            }
            break;
        }
    }
}
break;
default:
cout << "輸入錯誤，請重新輸入。" << endl;
}
```

```
//check monster die
if (!check_die()) {
    cout << "Monster die." << endl;
    break;
}
```

```
//畫面整理
cout << endl << endl;
//遊俠閃避攻擊
if (player->get_profession() == "遊俠" && rand()%3 == 0) {
    cout << "成功閃避攻擊" << endl;
    round++;
    continue;
}
```

```
//Monster attack
cout << get_name() << "發動攻擊" << endl;
player->cause_damage(get_atk());
//設定蜘蛛中毒
if (get_name() == "蜘蛛") {
    player->set_poison(2);
    cout << "受到蜘蛛攻擊，角色中毒" << endl;
}
```

```
//check player die
if (!player->check_die()) {
    cout << "You die." << endl;
    break;
}
```

```
//MP回復
int recover_value;
if (player->get_cur_hunger() < player->get_max_hunger() / 2) {
    recover_value = player->get_max_mp() / 10;
    cout << "飢餓中 ";
}
else {
    recover_value = player->get_max_mp() / 5;
}
```

```
//回合結束
cout << "回合結束 : " << endl;
round++;
//檢測HP增加後是否超過最大值
if ((player->get_cur_mp() + recover_value) >= player->get_max_mp()) {
    cout << "自然回復 " << player->get_max_mp() - player->get_cur_mp() << "點 MP" << endl;
    player->set_cur_mp(player->get_max_mp());
}
```

角色攻擊後，檢查怪獸是否死亡

```
else {
    cout << "自然回復 " << recover_value << "點 MP" << endl;
    player->set_cur_mp(player->get_cur_mp() + recover_value);
}
//CD時間減少
for (int i = 0; i < player->get_skills().size(); i++) {
    if (player->get_skills()[i]->get_cur_CD() != 0) {
        player->get_skills()[i]->CD_decrease(1);
    }
}
//畫面整理
cout << endl;
system("pause");
cout << endl;
}
```

```
cout << "戰鬥結束" << endl;
//玩家獲得金錢
int random_number = rand() % 30 + 10;
player->add_money(random_number);
//野豬設定(打倒野豬獲得豬肉)
```

```
Food* temp;
if (get_name() == "野豬") {
    temp = new Food("豬肉", 4);
    player->add_item(temp);
}
```

```
//畫面整理
cout << endl << endl;
system("pause");
system("cls");
```

怪獸攻擊後，檢查角色是否死亡

戰鬥結束後  
獲得的資源  
與物品

# 7. NPC

```
// Base Class
class NPC : public GameCharacter {
public:
    NPC();
    NPC(string, string, vector<Item*>);
    void list_commodity();
    virtual bool trigger_event(Object*) override;
    //setter
    void set_script(string);
    void set_commodity(vector<Item*>);
    //getter
    string get_script();
    vector<Item*> get_commodity();
    //erase item
    void erase_item(int);
private:
    string script;
    vector<Item*> commodity;
};
```

```
//玩家遭遇進行交易
class Businessman : public NPC {
public:
    Businessman();
    Businessman(vector<Item*>,vector<int>);
    bool trigger_event(Object*) override;
    void show_goods();
    //setter
    void set_cost(vector<int>);
    //getter
    vector<int> get_cost();
private:
    vector<int> cost;
};
```

```
//設定出現在 Start Room 紿予玩家基本物資
class Leader :public NPC {
public:
    Leader();
    bool trigger_event(Object*) override;
    void set_get_weapon(bool);
    bool get_get_weapon();
private:
    bool get_weapon;
};
```

```
//玩家遭遇可以學習一項 skill
class Teacher :public NPC {
public:
    Teacher();
    Teacher(vector<Skill*> );
    bool trigger_event(Object* ) override;
    void set_skills(vector<Skill*> );
    vector<Skill*>& get_skills();
private:
    vector<Skill*>skills;
    bool learn_skill;
};
```

以 NPC class 為基底，其他 class 繼承 NPC 並分別實作不同的功能。

virtual trigger\_event(Object\*)  
來進行與玩家相遇時，會觸發不同情況的設計。

# 7. NPC

## (一) Businessman： 負責與玩家交易相關功能

販賣的商品有武器與食物等，  
武器類只能購買一次，而其  
他種類則可以多次購買。

用 cost 儲存每項商品的價錢，  
並在購買時，判斷是否移除。

用 show\_goods() 列出商品，  
再讓玩家選擇要購買的商品，  
並加入金錢系統的判斷，來  
完成交易。

```
bool Businessman::trigger_event(Object* object) {
    Player* player = dynamic_cast<Player*>(object);
    //player shop
    int choice;
    while (1) {
        show_goods();
        cout << "金錢：" << player->get_money() << endl;
        cout << "請輸入想購買的物品(-1返回)：" ;
        cin >> choice;
        if (choice == -1) { break; }
        else if (choice <= 0 || choice > get_commodity().size()) {
            cout << "輸入錯誤，請重新輸入。" << endl;
        }
        else {
            if (player->get_money() < cost[choice - 1]) {
                cout << "金錢不足，交易失敗" << endl << endl;
            }
            else {
                player->add_item(get_commodity()[choice - 1]);
                player->set_money(player->get_money() - cost[choice - 1]);
                cout << "交易成功，花費" << cost[choice - 1] << "元，剩餘" << player->get_money() << "元。" << endl;
                if (get_commodity()[choice - 1]->get_tag() == "Weapon") {
                    erase_item(choice - 1);
                    cost.erase(cost.begin() + choice - 1);
                }
                cout << "" << endl;
            }
        }
    }
    cout << "謝謝惠顧，歡迎再次光臨。" << endl;
    return 1;
}
```

# 7. NPC (一) Businessman

依據不同商品做出不同的輸出，利用商品的 tag 來偵測。

```
void Businessman::show_goods() {
    //Debug
    if (cost.size() != get_commodity().size()) { cout << "Error(show_goods)" << endl; exit(1); }
    //
    cout << get_name() << ":" << get_script() << endl;
    //展示商品
    for (int i = 0; i < get_commodity().size(); i++) {
        if (get_commodity()[i]->get_tag() == "Weapon") {
            cout << i + 1 << ". " << "類別: 武器 \t" << get_commodity()[i]->get_name() << "\t\t / 花費: " << cost[i] << "\t / HP: " << get_commodity()[i]->get_hp();
        }
        else if (get_commodity()[i]->get_tag() == "Recover") {
            cout << i + 1 << ". " << "類別: 藥水 \t" << get_commodity()[i]->get_name() << "\t / 花費: " << cost[i] << "\t / 恢復數值: " << get_commodity()[i]->get_recover();
        }
        else if (get_commodity()[i]->get_tag() == "Food") {
            cout << i + 1 << ". " << "類別: 食物 \t" << get_commodity()[i]->get_name() << "\t\t / 花費: " << cost[i] << "\t / 恢復數值: " << get_commodity()[i]->get_recover();
        }
        else {
            cout << i + 1 << ". " << "類別: 無 \t" << get_commodity()[i]->get_name() << endl;
        }
    }
}
```

# 7. NPC

## (二) Teacher： 玩家學習技能相關功能

會提供兩種技能給玩家選擇想要學習的技能，一位師傅只能向他學習一次。

玩家觸發 trigger\_event() 先偵測是否學習過(利用learn\_skill判斷)，在顯示出可學習技能的名稱。判斷角色的技能數量，角色是否需要放棄技能。並透過角色的 GameCharacter::learn\_skill() 來將技能加入。學習完後將learn\_skill 設定成1。

```
bool Teacher::trigger_event(Object* object) {
    //轉換 object 到 gamecharacter
    GameCharacter* player = dynamic_cast<GameCharacter*>(object);

    //輸出基本資訊
    cout << get_name() << ":" << endl << get_script() << endl;

    //偵測玩家是否學習過 Skill
    if (learn_skill == 1) {
        cout << "你已經學習過技能。" << endl;
        return 0;
    }

    //輸出可以學習的招式
    cout << "招式名稱 / 消耗MP / 傷害 / CD " << endl;
    for(int i=0;i<skills.size();i++){
        cout << i + 1 << ". " << skills[i]->get_name() << " / " << skills[i]->get_cost() << " ,
    }

    //玩家選擇學習技能
    int choice;
    while (1) {
        cout << "請輸入想學習的招式(輸入-1返回): ";
        cin >> choice;
        if (choice == -1) { break; }
        else if (choice <= 0 || choice > skills.size()) {
            cout << "輸入錯誤，請重新輸入。" << endl;
        }
        else {
            player->learn_skill(skills[choice - 1]);
            skills.erase(skills.begin() + choice - 1);
            learn_skill = 1;
            break;
        }
    }
    return 1;
}
```

# 7. NPC

## (三) Leader :

給予玩家初始武器及金錢。

會與玩家達成一些對話，並給予玩家武器，Leader 擁有三把武器，而玩家只能從中選一把(利用get\_weapon來判斷)。當玩家選擇武器後，便將get\_weapon設定成 1。

```
void NPC::list_commodity(){
    if (commodity.size() != 0) {
        for (int i = 0; i < commodity.size(); i++) {
            if (get_commodity()[i]->get_tag() == "Weapon") {
                cout << i + 1 << ". " << "類別: 武器 \t" << get_commodity()[i]->get_name()
            }
            else if (get_commodity()[i]->get_tag() == "Recover") {
                cout << i + 1 << ". " << "類別: 藥水 \t" << get_commodity()[i]->get_name()
            }
            else if (get_commodity()[i]->get_tag() == "Food") {
                cout << i + 1 << ". " << "類別: 食物\ t" << get_commodity()[i]->get_name()
            }
        }
    } else {
        cout << "No commodity." << endl;
    }
}
```

NPC::list\_commodity()

村長：  
冒險者阿！  
現在我手上有一個探索地牢的任務，這個任務就交給你處理了。順帶一提，你沒有拒絕的權利。  
加油吧！冒險者解放地牢的任務就交給你了。  
這是我給你的一些資助，加油！(-\_-)

1. 類別：武器 大劍 / HP: 0 / ATK: 30 / DEF: 10  
2. 類別：武器 武士刀 / HP: 0 / ATK: 30 / DEF: 10  
3. 類別：武器 Excalibur / HP: 0 / ATK: 30 / DEF: 10  
請選擇初始武器(只能選擇一次)

↑ 執行結果

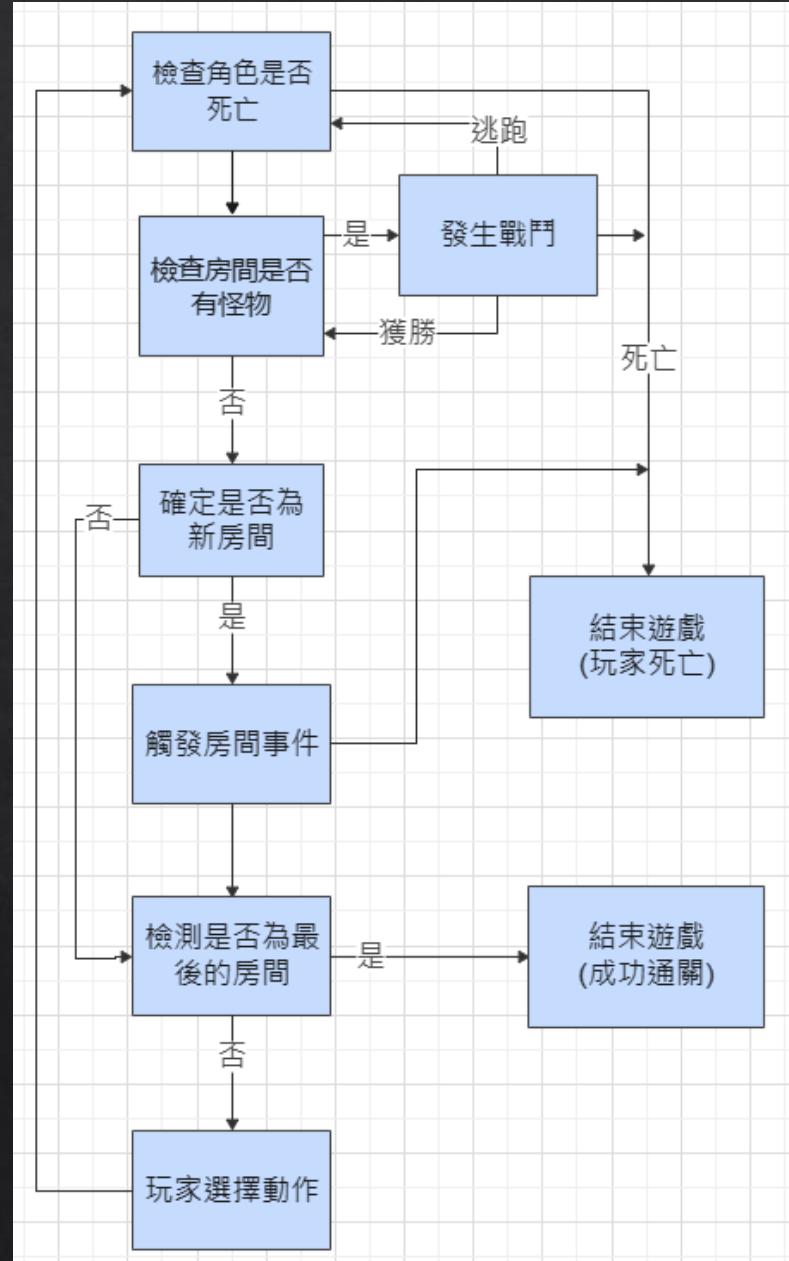
```
bool Leader::trigger_event(Object* object) {
    //
    if (get_weapon) {
        cout << "加油吧！勇者！我相信你能打敗迷宮之王的！(>_<)" << endl;
        return 1;
    }
    //
    Player* player = dynamic_cast<Player*>(object);
    cout << get_name() << ":" << endl;
    cout << get_script() << endl;

    //give money to player
    player->set_money(100); //initial money setting

    int choice;
    while (1) {
        if (get_weapon) {
            cout << "你已獲得初始武器，開始你的地城冒險吧！" << endl;
            break;
        }
        list_commodity();
        cout << "請選擇初始武器(只能選擇一次)" << endl;
        cin >> choice;
        if (choice<=0 || choice>get_commodity().size()) {
            cout << "輸入錯誤，請重新輸入。" << endl;
        }
        else {
            player->add_item(get_commodity()[choice - 1]);
            get_weapon = 1;
        }
    }
    system("Pause");
    system("cls");
    return 1;
}
```

# 8. Game Logic

整個遊戲的進行主要是在 `run_dungeon()` 中執行，其中運用到 `handle_event()`、`check_monster_room()` 等函式負責處理偵測，與執行特定事件的功能。最後，在遊戲結束時依據不同狀況呼叫，`show_game_end()` 或是 `show_game_over`。



# 8. Game Logic

```
void Dungeon::run_dungeon() {
    int choice;
    int go_to_new_room=1;
    Player* player=start_game();
    while (1) {
        system("cls");
        if (!player->check_die()) {
            show_game_over();
        }
        //1. trigger event
        handle_event(player);
        // room trigger event
        if (go_to_new_room) {
            player->get_current_room()->trigger_event(player);
            system("cls");
        }
        //2. check game end
        if (check_game_logic(player) && !check_monster_room(player->get_current_room())) {
            cout << "請選擇是否離開:\n1: 是\n2: 否" << endl;
            cin >> choice;
            if (choice == 1) {
                break;
                system("cls");
            }
        }
        //3. choose action
        go_to_new_room = choose_action(player, player->get_current_room()->get_objects());
        cout << endl;
        system("Pause");
    }
    show_game_end();
}
```

```
bool Dungeon::handle_retreat(Player* player) {
    char input;
    Room* temp;
    while (1) {
        Monster* monster = dynamic_cast<Monster*>(player->get_current_room()->get_objects()[0]);
        monster->show_status();
        cout << "請輸入是否撤退:" << endl << "1. 是" << endl << "2. 否" << endl;
        cin >> input;
        switch (input) {
        case '1':
            cout << "成功逃跑" << endl;
            player->change_room(player->get_previous_room());
            return 1;
            break;
        case '2':
            cout << "繼續戰鬥" << endl;
            return 0;
            break;
        default:
            cout << "輸入錯誤，請重新輸入。" << endl;
        }
    }
}
```

```
bool Dungeon::check_game_logic(Player* player) {
    if (player->get_current_room()->get_is_exit()) {
        return 1;
    }
    return 0;
}
```

```
bool Dungeon::check_monster_room(Room* room ) {
    for (int i = 0; i < room->get_objects().size(); i++) {
        if (room->get_objects()[i]->get_tag() == "Monster") {
            return 1;
            break;
        }
    }
    return 0;
}
```

# 8. Game Logic

負責處理，房間裡有 Monster 的狀況。先詢問是否撤退，若不撤退，就用 trigger\_event() 觸發 Monster 的戰鬥系統。最後，若角色死亡，呼叫 show\_game\_over()。若怪獸死亡，將怪獸從房間刪除。

-----  
獅子  
HP: 50/50  
ATK: 20  
DEF: 10  
-----

請輸入是否撤退：  
1. 是  
2. 否

```
void Dungeon::handle_event(Player* player) {
    int check_retreat = 1;
    //偵測是否為Monster Room
    while (1) {
        if (check_monster_room(player->get_current_room())) {
            //逃跑
            check_retreat = handle_retreat(player);
        }
        else {
            //沒有Monster
            break;
        }
        if (!check_retreat) {
            system("cls");
            //戰鬥
            if (player->get_current_room()->get_objects()[0]->trigger_event(player)) {
                //Monster 死亡
                player->get_current_room()->get_objects().erase(player->get_current_room()->get_objects().begin());
            }
            else {
                //Player 死亡
                if (!player->check_die()) {
                    show_game_over();
                }
                break;
            }
        }
        else { break; } //已逃跑
    }
}
```

# 9. Hunger System

狀態設定：

- 1) Hunger (飢餓值)：當小於一半時，使戰鬥中自然回復MP減半。
- 2) Thirst (口渴值)：當小於等於 3 時，造成戰鬥中無法使用技能。
- 3) Poison (中毒值)：會在行動時造成損血
- 4) Power (活力值)：當小一半時，會讓攻擊下降 $1/3$ ，當歸零時，就無法移動。

#攻擊下降、MP回復、無法使用技能的偵測，於戰鬥系統部分說明。

```
class Player :public GameCharacter
private:
int max_thirst;
int cur_thirst; //當歸零時，造成戰鬥中無法使用技能
int max_hunger;
int cur_hunger; //當小於一半時，使戰鬥中自然回復MP減半
int max_power;
int cur_power; //小於  $1/2 * \text{max\_power}$ ：攻擊力下降；等於 0：無法移動
int poison; // 0: no poison 1: cause by swamp 2: cause by monster
```

```
bool Player::thirst_hunger_system() {
    bool flag1 = check_hunger();
    bool flag2 = check_thirst();
    bool flag3 = check_power();
    return (flag1 || flag2 || flag3);
}
```



負責處理狀態的成員函式，最後回傳角色是否死亡。

# 9. Hunger System

## (1) Hunger (飢餓值) :

消耗機制:

移動時依照地區消耗不同點數，Forest  
2點、其他地區 1 點。

回復機制:

透過吃下食物回復，食物獲得來源：  
商人販賣、打倒怪物(野豬)、  
環境 (Desert、Forest)

先判定現在飢餓值是否為0，如果小於0，  
減少HP並回傳角色是否死亡。如果大  
於0，減少 cur\_hunger，在 Forest 時，  
減少加倍。

```
bool Player::check_hunger() {  
    if (cur_hunger == 0) {  
        cout << "正在飢餓" << endl;  
        set_cur_hp(get_cur_hp() - 1);  
        return check_die();  
    }  
    else {  
        if (current_room->get_name() == "Forest") {  
            cur_hunger -= 2;  
            cout << "Hunger(飢餓值) 減少 2 點" << endl;  
        }  
        else {  
            cur_hunger--;  
            cout << "Hunger(飢餓值) 減少 1 點" << endl;  
        }  
        if (cur_hunger < 0) { cur_hunger = 0; }  
        return 1;  
    }  
}
```

# 9. Hunger System

## (2) Thirst (口渴值) :

消耗機制:

移動時依照地區消耗不同點數，Desert  
2點、其他地區 1 點。

回復機制:

透過飲用房間內的泉水回復，吃下仙人  
掌，或是和商人交易取得水後再飲用。

先判定現在口渴值是否為0，如果小於0，  
減少HP並回傳角色是否死亡。如果大於  
0，減少cur\_thirst，在 Desert 時，減少  
加倍。



```
bool Player::check_thirst() {
    if (cur_thirst == 0) {
        cout << "正在口渴" << endl;
        set_cur_hp(get_cur_hp() - 1);
        return check_die();
    }
    else {
        if (current_room->get_name() == "Desert") {
            cur_thirst -= 2;
            cout << "Thirst(口渴值) 減少 2 點" << endl;
            if (cur_thirst < 0) { cur_thirst = 0; }
        }
        else {
            cur_thirst--;
            cout << "Thirst(口渴值) 減少 1 點" << endl;
        }
    }
    return 1;
}
```

# 9. Hunger System

## (3) Poison (中毒值) :

依據不同中毒狀態扣除，不同血量。

中毒效果：

poison=0 沒有中毒

poison=1 減少1/10的最大HP

poison=2 減少1/5的最大HP

中毒狀態觸發：

poison=1 在Swamp中觸發事件

Poison=2 在森林中受到蜘蛛攻擊

中毒狀態解除：

方法一：喝下牛奶(商人販售)

方法二：使用解毒草(在森林中獲得)

```
bool Player::poison_system() {
    switch (poison)
    {
        case 0: //no poison
            break;
        case 1:
            set_cur_hp(get_cur_hp() - get_max_hp() / 10); // 扣1/10 的最大血量
            cout << "中毒損失 " << get_max_hp() / 10 << "點 HP" << endl;
            return check_die();
            break;
        case 2:
            set_cur_hp(get_cur_hp() - get_max_hp() / 5); // 扣 1/5 的最大血量
            cout << "中毒損失 " << get_max_hp() / 5 << "點 HP" << endl;
            return check_die();
            break;
        default:
            cout << "poison_system error. poison:" << poison << endl;
            exit(1);
            break;
    }
    return 1;
}
```

# 9. Hunger System

## (3) Poison (中毒值) :

### 中毒狀態觸發(一)

```
bool Swamp::trigger_event(Object* object) {
    Player* player = dynamic_cast<Player*>(object);
    if (type == 0 && !get_event_happen()) {
        cout << "觸碰到有毒植物，玩家中毒" << endl;
        player->set_poison(1);
        set_event_happen(1);
        cout << endl;
        system("pause");
        cout << endl;
    }
}
```

### 中毒狀態觸發(二)

```
void Monster::fighting_system(Player* player)
    //Monster attack
    cout << get_name() << "發動攻擊" << endl;
    player->cause_damage(get_atk());
    //設定蜘蛛中毒
    if (get_name() == "蜘蛛") {
        player->set_poison(2);
        cout << "受到蜘蛛攻擊，角色中毒" << endl;
    }
}
```

中毒狀態解除: 透過偵測玩家使用物品的 name 來判斷是否為牛奶或解毒草

```
void Player::use_item(int pos)
else if (inventory[pos]->get_name() == "牛奶" || inventory[pos]->get_name() == "解毒草") {
    poison = 0;
    cout << " 使用 " << inventory[pos]->get_name() << endl;
    cout << "成功解毒" << endl;
}
```

# 9. Hunger System

## (4) Power (活力值) :

消耗機制:

在移動房間時會消耗 1 點，但角色如果位於 Swamp 時，消耗兩點。

恢復機制:

角色能夠在選擇動作時，選擇「休息」，可以回復活力。當活力小於一半時，回復 2 點。當大於一半時，則回復一點。

特殊效果:

當 power 值為 0 時，角色則無法移動。

判定 power 是否為 0，在決定是否會造成損血。偵測玩家的 cur\_room 的 name 是否為 Swamp。再依據結果判斷要減少了 power 。

```
bool Player::check_power() {
    if (cur_power == 0) {
        cout << "感到疲勞" << endl;
        set_cur_hp(get_cur_hp() - 1);
        return check_die();
    }
    else {
        if (current_room->get_name() == "Swamp") {
            set_cur_power(get_cur_power() - 2);
            cout << "Power(活力值) 減少 2 點" << endl;
        }
        else {
            set_cur_power(get_cur_power() - 1);
            cout << "Power(活力值) 減少 1 點" << endl;
        }
    }
    if (get_cur_power() < 0) {
        set_cur_power(0);
    }
    return 1;
}
```



# 9. Hunger System

## (4) Power (活力值) :

特殊機制:

在選擇移動動作時，先偵測 power 值是否為 0。來達成活力不足無法移動的效果。

```
bool Dungeon::choose_action(Player* player, vector<Object*> objects)
{
    case 3: //角色移動
        if (player->get_cur_power() == 0) {
            cout << "活力不足，無法移動" << endl;
        }
        else {
            cout << endl;
            handle_movement(player);
            return 1;
        }
    break;
```

回復機制:

偵測現在的 power 是否低於一半。  
並依造偵測結果，回復對應的值。  
最後，如果超過最大值，就將  
cur\_power 設成最大值。

```
void Player::take_break() {
    check_hunger();
    check_thirst();
    poison_system();
    if (cur_power < max_power / 2) {
        cur_power += 2;
        cout << "回復 2 點活力" << endl;
    }
    else {
        cur_power += 1;
        cout << "回復 1 點活力" << endl;
    }
    if (cur_power > max_power) {
        cur_power = max_power;
    }
}
```

# 9. Hunger System

食物設定：

仙人掌：同時回復 Hunger 和 Thirst

解毒草、牛奶：解毒

水：回復 Thirst

其他食物：回復 Hunger

獲得方式：

(一)商人販賣：

香蕉、雞腿、水、牛奶

(二)房間特殊事件：

解毒草：在 Forest 中有機會獲得

蘋果：在 Forest 中有機會獲得

豬肉：在 Forest 中打倒野豬後獲得

仙人掌：在 Desert 中有機會採集

```
case 0:  
    temp = new Food("蘋果", 2);  
    break;  
case 1:  
    temp = new Food("香蕉", 2);  
    break;  
case 2:  
    temp = new Food("雞腿", 4);  
    break;  
case 3:  
    temp = new Food("水", 4);  
    break;  
case 4:  
    temp = new Food("牛奶", 1);  
    break;  
case 5:  
    temp = new Food("豬肉", 4);  
    break;  
case 6: // 可以同時回復 thirst 和 hunger  
    temp = new Food("仙人掌", 2);  
    break;  
case 7:  
    temp = new Food("解毒草", 1);  
    break;
```

# 10. Room System Design

Area.h 的檔案中宣告 Desert 、 Forest 、 Swamp 、 Start\_room 、 Final\_room 等 class 繼承至 Room ，負責裡房間中的地區功能。

1. Desert

```
class Desert :public Room {  
public:  
    Desert(bool, int, vector<Object*>);  
    Desert();  
    Desert(int);  
    bool trigger_event(Object*) override;  
  
    //getter & setter  
    int get_type();  
    void set_type(int);  
  
private:  
    int type;  
};
```

2. Forest

```
class Forest :public Room {  
public:  
    Forest(bool, int, vector<Object*>);  
    Forest();  
    Forest(int);  
    bool trigger_event(Object*) override;  
  
    //getter & setter  
    int get_type();  
    void set_type(int);  
  
private:  
    int type;  
};
```

3. Swamp

```
class Swamp :public Room {  
public:  
    Swamp(bool, int, vector<Object*>);  
    Swamp();  
    Swamp(int);  
    bool trigger_event(Object*) override;  
  
    //getter & setter  
    int get_type();  
    void set_type(int);  
  
private:  
    int type;  
};
```

4. Start\_room

```
class Start_room : public Room {  
public:  
    Start_room();  
    void start_room_setting();  
    bool trigger_event(Object*) override;  
  
private:  
};
```

5. Final\_room

```
class Final_room :public Room {  
public:  
    Final_room(int);  
    void final_room_setting();  
    bool trigger_event(Object*) override;  
  
private:  
};
```

# 10. Room System Design

## Desert

1. Oasis : 回復Thirst
2. Sandstroms : 減少Thirst & Hunger
3. 仙人掌 : 補充Thirst & Hunger
4. None : 空房間

## Forest

1. Lake :回復Thirst
2. 解毒草 :解毒
3. 水果 :回復Hungerr
4. 蜘蛛 :使玩家中毒
5. None :

```
void Desert::trigger_event(GameObject* object) {
    Player* player = dynamic_cast<Player*>(object);
    int choice;
    cout << "現在地區 : Desert" << endl << endl;
    if (type==1) {
        //oasis
        //輸出綠洲資訊
        cout << "在房間中有綠洲。" << endl;
        cout << "是否飲用綠洲泉水(回復Thirst值):" << endl;
        cout << "1. 是" << endl;
        cout << "2. 否" << endl;
        //Player 選擇
        cin >> choice;
        if (choice == 1) {
            player->set_cur_thirst(player->get_max_thirst());
            cout << "飲用泉水回復 Thirst 至最大值" << endl;
        }
        else { cout << "離開綠洲" << endl; }
        cout << endl;
        system("pause");
        cout << endl;
    }
    else if(type == 0 && !get_event_happen()){
        //sandstorms
        cout << "遭遇沙塵暴，飢餓值與水分大量減少。(減少最大值的1/3)" << endl;
        //水分與飢餓值減少1/3
        player->set_cur_thirst(player->get_cur_thirst() - player->get_max_thirst() / 3);
        player->set_cur_hunger(player->get_cur_hunger() - player->get_max_hunger() / 3);
        //檢查 Hunger & Thirst 是否小於 0
        if (player->get_cur_hunger() < 0) {
            player->set_cur_hunger(0);
        }
        if (player->get_cur_thirst() < 0) {
            player->set_cur_thirst(0);
        }
        //
        cout << " Thirst : " << player->get_cur_thirst() << " / " << player->get_max_thirst() << endl;
        cout << " Hunger : " << player->get_cur_hunger() << " / " << player->get_max_hunger() << endl;
        //畫面整理
        cout << endl;
        system("pause");
        cout << endl;
        //設定事件已發生
        set_event_happen(1);
    }
    else if (type == 2 && !get_event_happen()) {
        cout << endl << endl;
        cout << "在沙漠中發現仙人掌" << endl;
        Food* temp =new Food("仙人掌", 2);
        player->add_item(temp);
        //設定事件已發生
        set_event_happen(1);
    }
    return 1;
}
```

# 10. Room System Design

## Swamp

區域設定:

1. hunger & thirst 消耗加倍
2. 遭受水蛭攻擊，機率性扣血 (type = 1)
3. 觸碰到有毒植物，中毒 (type = 0)
4. 乾淨水源 (type = 2)
5. 發現武器(type = 3)

運用virtual function 的概念來執行地區功能。Payer 儲存是以Room\*的型態儲存資料，所以透過 virtual function 能讓每個地區在呼叫trigger\_event() 時，能夠做出不同的事件。

```
class Object
virtual bool trigger_event(Object*) = 0;

bool Room::trigger_event(Object* player) { return 0; }
```

```
bool Swamp::trigger_event(Object* object) {
    Player* player = dynamic_cast<Player*>(object);
    cout << "現在地區 : Swamp" << endl << endl;
    if (type == 0 && !get_event_happen()) {
        cout << "觸碰到有毒植物，玩家中毒" << endl;
        player->set_poison(1);
        set_event_happen(1);
        cout << endl;
        system("pause");
        cout << endl;
    }
    else if(type == 1 && !get_event_happen()) {
        cout << "房間事件" << endl;
        cout << "遭受水蛭攻擊，損失血量" << endl;
        player->cause_damage(player->get_max_hp()/5 + player->get_def()); //損失 1/5 最大值血量
        set_event_happen(1);
        cout << endl;
        system("pause");
        cout << endl;
    }
    else if (type == 2) {
        int choice;
        cout << "在房間中有乾淨水源。" << endl;
        cout << "是否飲用水源(回復Thirst值):" << endl;
        cout << "1. 是" << endl;
        cout << "2. 否" << endl;
        cin >> choice;
        if (choice == 1) {
            player->set_cur_thirst(player->get_max_thirst());
            cout << "飲用水源回復 Thirst 至最大值" << endl;
        }
        else {
            cout << "離開水源" << endl;
        }
        cout << endl;
        system("pause");
        cout << endl;
    }
    else if (type == 3) {
        cout << "發現破舊小屋" << endl;
        cout << "進去探索後，發現武器" << endl;
        Weapon* temp = new Weapon("Excalibur", 0, 30, 10);
        player->add_item(temp);
    }
    return player->check_die();
}
```

```
bool Forest::trigger_event(Object* object) {
    Player* player = dynamic_cast<Player*>(object);
    int choice;
    cout << "現在地區 : Forest" << endl << endl;
    if (type == 0) {
        cout << "在房間中有湖泊。" << endl;
        cout << "是否飲用湖水(回復Thirst值):" << endl;
        cout << "1. 是" << endl;
        cout << "2. 否" << endl;
        cin >> choice;
        if (choice == 1) {
            player->set_cur_thirst(player->get_max_thirst());
            cout << "飲用湖水回復 Thirst 至最大值" << endl;
        }
        else {
            cout << "離開湖邊" << endl;
        }
        cout << endl;
        system("pause");
        cout << endl;
    }
    else if (type == 1 && !get_event_happen()) {
        Food* temp=new Food("解毒草", 1);
        cout << "在森林中找到解毒草" << endl;
        player->add_item(temp);
        set_event_happen(1);
        cout << endl;
        system("pause");
        cout << endl;
    }
    else if (type == 2 && !get_event_happen()) {
        Food* temp=new Food("蘋果", 2);
        cout << "在樹上發現蘋果" << endl;
        player->add_item(temp);
        cout << endl;
        set_event_happen(1);
    }
    return 1;
}
```

# 10. Room System Design

## Start\_room

設定村長會出現在這個房間，當玩家進入房間時，回與村長互動。

## Final\_room

Boos 出現的房間，在打倒Boss 玩家可以選擇離開地下城，成功通關遊戲。

```
void Start_room::start_room_setting() {
    set_index(0);
    Leader* leader=new Leader;
    vector<Object*> objects;
    objects.push_back(leader);

    //obects
    set_objects(objects);
}

bool Start_room::trigger_event(Object* object) {
    Player* player = dynamic_cast<Player*>(object);
    get_objects()[0]->trigger_event(player);
    return 1;
}

void Final_room::final_room_setting(){
    Boss* boss = new Boss();
    vector <Object*> objects;
    objects.push_back(boss);
    set_objects(objects);
    set_is_exit(1);
}

bool Final_room::trigger_event(Object* object) {
    Player* player = dynamic_cast<Player*>(object);
    cout << "Boss Room" << endl;

    return 1;
}
```

# Optional Enhancement

- 1. 技能系統
- 2. MP
- 3. CD
- 4. 職業系統
- 5. 遊戲難度選擇
- 6. 玩家地圖
- 7. 金錢系統

# 1. 技能系統

玩家在戰鬥中可以使用技能，並造成較高的傷害。而使用技能需要消耗MP，且在使用後會有CD時間。

傷害計算：造成傷害 = 角色傷害 + 技能傷害

技能取得：

玩家在一開始選擇職業時，每個職業都會分配不同的專屬技能。此外在地下城中，有出現「師傅」（一種NPC）可以向他學習一種技能。

限制：

玩家有技能數量有限制（戰士、遊俠：2個；法師：3個）  
若技能已滿後再去學習技能，則要先捨去一個技能。

```
class Skill : public Object {
public:
    Skill();
    Skill(string name,int cost,int damage,int CD);
    bool trigger_event(Object*) override;
    void CD_decrease(int);
    //getter & setter
    int get_cost();
    int get_damage();
    int get_CD();
    int get_cur_CD();
    void set_cost(int);
    void set_damage(int);
    void set_CD(int);
    void set_cur_CD(int);
private:
    int cost;      //consume mp
    int damage;    //cause damage
    int CD;        //Cooling Down Time
    int cur_CD;
};

Skill* Skill_setting(int type) {
    switch (type) {
    case 0:
        temp = new Skill("西瓜榴槤擊",10,30,3);
        break;
    case 1:
        temp = new Skill("The World!!!",15,40,4);
        break;
    case 2:
        temp = new Skill("昇龍拳",7,15,2);
        break;
    case 3:
        temp = new Skill("發炎拳", 7, 12,2);
        break;
    case 4: //戰士預設技能
        temp = new Skill("斬擊",6,5,2);
        break;
    case 5: //遊俠預設技能
        temp = new Skill("突刺", 12, 10, 2);
        break;
    case 6: //法師預設技能
        temp = new Skill("豪火球之術", 20, 35 , 4);
        break;
    }
}
```

全部種類的技能(名字,傷害,消耗MP,CD)

# 1. 技能系統

```
class GameCharacter : public Object{  
private:  
    int skill_number = 2;  
    vector<Skill*> skills;  
  
    temp->set_profession("法師");  
    temp->learn_skill(Skill_setting(6));  
    temp->set_skill_number(3);
```

技能儲存在 skills 中，skill\_number 是指最多可以擁有的技能數量。在法師的設定中，會將 skill\_number 設定為 3

```
int GameCharacter::use_skill(int num) {  
    if (num < 0 || num >= skills.size()) {  
        cout << "輸入數字錯誤(use_skill)" << endl;  
        exit(1);  
    }  
    cur_mp -= (skills[num]->get_cost()); MP消耗  
    cout << get_name() << " 使用 " << skills[num]->get_name() << endl;  
    skills[num]->set_cur_CD(skills[num]->get_CD()); CD時間設定  
    return skills[num]->get_damage() + atk;  
}
```

使用技能(傳回造成傷害的數值)

```
bool GameCharacter::learn_skill(Skill* new_skill) {  
    if (skills.size() == skill_number) { 判斷技能數量  
        cout << "請選擇要去棄的技能:" << endl;  
        int choice=-1;  
        show_skill();  
        while (1) {  
            cin >> choice;  
            if (choice <= 0 || choice > skills.size()) {  
                cout << "輸入錯誤，請重新輸入。" << endl;  
            }  
            else {  
                break;  
            }  
        }  
        skills.erase(skills.begin() + choice-1);  
        skills.push_back(new_skill);  
    }  
    else {  
        skills.push_back(new_skill);  
    }  
    cout << "Learn Skill Success" << endl;  
    show_skill();  
    return 1;  
}
```

技能學習：  
先偵測現在的技能數量，  
在判斷是否丟棄技能

## 2. MP系統

玩家選擇使用技能後，會在 `use_skill()` 中扣除使用技能的需消耗的MP。

戰鬥中，每回合結束會回復 MP最大值的 $1/5$ ，但當飢餓狀態發生時，變為回復最大值的 $1/10$ 。回復前先偵測玩家的 Hunger，並判斷是否到達飢餓狀態。

```
int GameCharacter::use_skill(int num) {
    if (num < 0 || num >= skills.size()) {
        cout << "輸入數字錯誤(use_skill)" << endl;
        exit(1);
    }
    cur_mp -= (skills[num]->get_cost());
    cout << get_name() << " 使用 " << skills[num]->get_name() << endl;
    return skills[num]->get_damage() + atk;
}
```

```
void Monster::fighting_system(Player* player)
{
    //MP 回復
    int recover_value;
    if (player->get_cur_hunger() < player->get_max_hunger() / 2) {
        recover_value = player->get_max_mp() / 10;
        cout << "飢餓中 ";
    }
    else {
        recover_value = player->get_max_mp() / 5;
    }
    if ((player->get_cur_mp() + recover_value) >= player->get_max_mp()) {
        cout << "自然回復 " << player->get_max_mp()-player->get_cur_mp() << "點 MP" << endl;
        player->set_cur_mp(player->get_max_mp());
    }
    else {
        cout << "自然回復 " << recover_value << "點 MP" << endl;
        player->set_cur_mp(player->get_cur_mp() + recover_value);
    }
}
```

# 3. CD 時間

玩家進入戰鬥時，將所有技能的CD時間設為0。玩家在戰鬥時，使用技能後會產生CD時間。CD時間在回合結束時減少。當CD時間為零時，便可再次使用技能。

```
void Skill::CD_decrease(int n) {
    cur_CD -= n;
    if (cur_CD <= 0) {
        cur_CD = 0;
        cout << "技能：" << get_name() << "準備完成" << endl;
    } else {
        cout << "技能：" << get_name() << "\tCD：剩餘 " << cur_CD << "回合" << endl;
    }
}
```

Skill::CD\_decrease(int n)  
減少 n 回合的CD時間

```
void Monster::fighting_system(Player* player) {
    //歸零CD時間
    for (int i = 0; i < player->get_skills().size(); i++) {
        player->get_skills()[i]->set_cur_CD(0);
    }

    //CD 時間限制玩家使用技能
    else if (player->get_skills()[choice1 - 1]->get_cur_CD() != 0) {
        cout << player->get_skills()[choice1 - 1]->get_name() << "：技能CD中，無法使用" << endl << endl;
    }

    //CD 時間減少
    for (int i = 0; i < player->get_skills().size(); i++) {
        if (player->get_skills()[i]->get_cur_CD() != 0) {
            player->get_skills()[i]->CD_decrease(1);
        }
    }
}
```

fighting\_system 中處理 CD 時間的相關程式

# 4. 職業系統

在 Player\_setting() 中設定各個職業的數值及技能，並在 create\_player() 中提供玩家選擇職業。

```
Player* Dungeon::create_player()
{
    cout << "-----職業選擇-----" << endl;
    cout << "    職業：角色特性" << endl;
    cout << "1. 戰士：高血量 防禦高 攻擊力低" << endl;
    cout << "2. 遊俠：血量低 防禦低 攻擊力高 有機會閃避攻擊" << endl;
    cout << "3. 法師：血量低 防禦低 攻擊力最高 MP多" << endl;
    cout << "請選擇職業：" << endl;
    cin >> choice;

    player= Player_setting(choice - 1);
```

```
//player setting
Player* Player_setting(int type) {
    Player* temp=NULL;
    switch (type) {
        case 0:
            temp = new Player("", 70, 15, 20 , 15);
            temp->set_profession("戰士");
            temp->learn_skill(Skill_setting(4));
            break;
        case 1:
            temp = new Player("", 60, 20, 10, 20);
            temp->set_profession("遊俠");
            temp->learn_skill(Skill_setting(5));
            break;
        case 2:
            temp = new Player("", 55, 25, 10,35);
            temp->set_profession("法師");
            temp->learn_skill(Skill_setting(6));
            temp->set_skill_number(3);
            break;
    }
}
```

## 4. 職業系統

在Player 中有變數 profession 儲存角色職業。而在戰鬥系統中，遊俠特性是有機率閃避攻擊，我利用rand()隨機生成數值，並取其餘數，來達成機率閃避攻擊的效果。

```
class Player :public GameCharacter {  
    string profession;  
  
    //遊俠閃避攻擊  
    if (player->get_profession() == "遊俠" && rand()%3 == 0) {  
        cout << "成功閃避攻擊" << endl;  
        round++;  
        continue;  
    }
```

# 5. 難度設定

在 Dungeon 中設有 private 的陣列，儲存各個難度需要的資料。在遊戲開始時，玩家輸入難度的選擇，並儲存在 mode 中。最後依照儲存資料與玩家輸入的資訊生成地圖。

實際執行結果 →

```
-----難度選擇-----  
1. 簡單 : 商人多 怪物少 房間數量少  
2. 中等 : 商人多 怪物多 房間數量中  
3. 困難 : 商人少 怪物多 房間數量多  
2  
a 難度設定中等
```

```
void Dungeon::create_map(Player* player, int area_number)  
{  
    //GameCharacter setting  
    int b_number = businessman_number[mode];  
    int t_number = teacher_number[mode];  
    int m_number = monster_number[mode];  
    int number_gamecharacter = b_number + t_number + m_number; //NPC number setting
```

```
Player* Dungeon::start_game()  
cout << "-----難度選擇-----" << endl;  
cout << "1. 簡單 : 商人多 怪物少 房間數量少" << endl;  
cout << "2. 中等 : 商人多 怪物多 房間數量中" << endl;  
cout << "3. 困難 : 商人少 怪物多 房間數量多" << endl;  
while (1) {  
    cin >> choice;  
    if (choice >= 1 && choice <= 3) { break; }  
    else { cout << "輸入錯誤" << endl; }  
}  
mode = choice - 1;
```

```
class Dungeon  
{  
    //難度  
    int mode;  
    //數量 Dictionary ( 簡單 , 中等 , 困難 )  
    int businessman_number[3] = { 3, 2, 1 };  
    int teacher_number[3] = { 2, 1, 1 };  
    int monster_number[3] = { 2, 4, 6 };  
    int room_number[3] = { 10, 12, 14 };
```

# 6. 玩家地圖

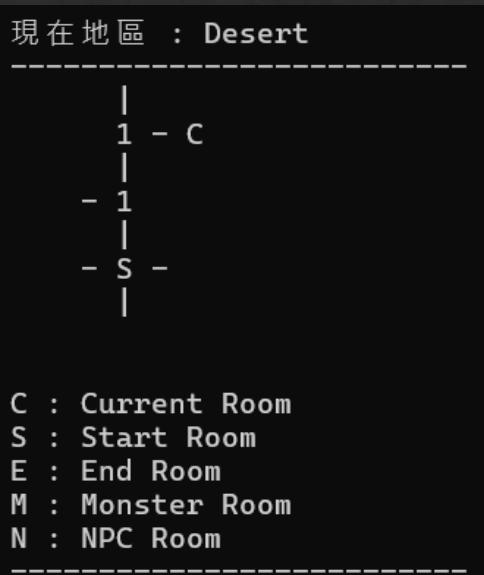
Map :

負責處理房間的資訊，在 Player 中有 Map 型態的資料，負責處理玩家走過的房間資訊。在生成地圖中，為了避免在同樣位置上初先不同的房間，也有利用 Map 紀錄已經生成過的房間。初始的位置設定在3\*3的正中心。

Room\* 的二維vector陣列，記錄下所有房間的位置。可以隨著地圖變大，而增加 map 的大小。

Show\_map() : 在 cmd 上顯示地圖，依據房間的內容顯示出不同圖示，且房間相連的通道也會一併顯示。

地圖顯示範例



```
class Map {
public:
    //Constructor
    Map();
    Map(int, int);
    //初始化地圖
    void initial_map();
    //在 map 中尋找 room
    Pos search(Room*);
    //設定room 在 pos 的位置上
    void set_map(Room*, Pos);
    //更新map資訊
    void update_map(Room*, Room*);
    //偵測在 Map 上周圍是否有房間
    vector<int> check_near_room(Pos);
    //顯示map
    void show_map();
    //getter & setter
    int get_size_x();
    int get_size_y();
    vector<vector<Room*>> get_map();
    void set_size_x(int);
    void set_size_y(int);
    void set_size(int, int);
private:
    //增加 map 的大小
    void increase_map_size(int, char);
    //地圖在 x 方向的大小
    int map_size_x;
    //地圖在 y 方向的大小
    int map_size_y;
    vector<vector<Room*>> map;
};
```

# 6. 玩家地圖

Show\_map() :

偵測每個位置上的Room\* 決定輸出的內容，並在輸出時判斷房間與四周的道路。偵測原則：每個房間都偵測右方(下方)道路，如果左方(上方)房間為NULL則偵測左方(上方)道路。

```
void Map::show_map() {
    cout << "-----" << endl;
    //偵測第一行上方道路
    cout << " ";
    for (int i = 0; i < map[0].size(); i++) {
        if (map[0][i] != NULL && map[0][i]->get_up_room() != NULL) {
            cout << "| ";
        }
        else {
            cout << " ";
        }
    }
    cout << endl;
    //show other map
    for (int i = 0; i < map.size(); i++) {
        for (int j = 0; j < map[i].size(); j++) {
            if (map[i][j] != NULL) {
                //偵測左房間
                if (j > 0 && (map[i][j - 1] == NULL || map[i][j - 1]->get_right_room() == NULL))
                    //偵測其他列
                    if (map[i][j]->get_left_room() != NULL) {
                        cout << "- ";
                    }
                    else {
                        if (map[i][j - 1] == NULL) {
                            cout << " ";
                        }
                    }
                else if (j == 0) {
                    //偵測第一列
                    if (map[i][j]->get_left_room() != NULL) {
                        cout << "- ";
                    }
                    else {
                        cout << " ";
                    }
                }
                //檢查房間種類
                if (map[i][j]->get_go_through() == 2) {
                    cout << "C "; //current room
                }
                else if (map[i][j]->get_name() == "Start_room") {
                    cout << "S "; //start room
                }
                else if (map[i][j]->get_name() == "Final_room") {
                    cout << "E "; //final room
                }
                else if (map[i][j]->get_objects().size() != 0 && map[i][j]->get_objects()[0]->get_tag() == "NPC") {
                    cout << "N "; //NPC room (expect start room)
                }
                else if (map[i][j]->get_objects().size() != 0 && map[i][j]->get_objects()[0]->get_tag() == "Monster") {
                    cout << "M "; //monster room (expect final room)
                }
                else if (map[i][j]->get_go_through() == 1) {
                    cout << "I ";
                }
                else {
                    cout << " ";
                }
                //偵測右房間
                if (map[i][j]->get_right_room() != NULL) {
                    cout << "- ";
                }
                else {
                    cout << " ";
                }
            }
            else {
                //map 上房間為 NULL
                if (j == 0) { cout << " "; }
                cout << " ";
                if (j == map[i].size() - 1 || map[i][j + 1] == NULL) {
                    cout << " ";
                }
            }
            cout << endl;
            //偵測上下房間
            cout << " ";
            for (int k = 0; k < map[i].size(); k++) {
                if (map[i][k] == NULL) {
                    cout << "-----" << endl;
                    cout << "C : Current Room" << endl;
                    cout << "S : Start Room" << endl;
                    cout << "E : End Room" << endl;
                    cout << "M : Monster Room" << endl;
                    cout << "N : NPC Room" << endl;
                    cout << "-----" << endl;
                }
                else if (map[i][k] != map[i][k + 1]) {
                    cout << "-----" << endl;
                    cout << "-----" << endl;
                }
            }
        }
    }
}
```

# 6. 玩家地圖

Map::update\_map() :

當角色移動房間時呼叫，需要船近兩個 Room\*，先判斷 previous\_room 和 current\_room 的相對位置關係，透過 Pos 儲存。在利用 search() 找到 previous\_room 在地圖上的位置，最後再將 current\_room 設定在地圖上。

Search\_room() :

輸入 Room\* 利用迴圈搜尋，將找到的座標以 Pos 的形式傳出。

```
Pos Map::search(Room* target) {
    Pos temp;
    for (int i = 0; i < map.size(); i++) {
        for (int j = 0; j < map[i].size(); j++) {
            if (target == map[i][j]) {
                temp = Pos(j, i);
            }
        }
    }
    //Debug
    if (temp.x == -1) { cout << "Not find room in map.\n"; exit(1); }
    return temp;
}
```

```
void Map::update_map(Room* current_room, Room* previous_room) {
    Pos relation, final_pos;
    if (previous_room != NULL) {
        switch (previous_room->find_relation(current_room)) {
            case 'U':
                relation = Pos(0, -1);
                //cout << "U" << endl;
                break;
            case 'D':
                relation = Pos(0, 1);
                //cout << "D" << endl;
                break;
            case 'R':
                relation = Pos(1, 0);
                //cout << "R" << endl;
                break;
            case 'L':
                relation = Pos(-1, 0);
                //cout << "L" << endl;
                break;
            default:
                cout << "Error Update map." << endl;
        }
        final_pos = search(previous_room) + relation;
        set_map(current_room, final_pos);
    }
}
```

# 7. 金錢系統

玩家與商人交易需要使用金錢，不同物品有不同的價錢。當金錢不足時，便無法成功交易。

金錢取得方法：

- (一)打倒怪物可以隨機獲得一定數量的金錢。
- (二)在遊戲開始，與村長對話可以獲得一定量的金錢。

實作：

在Player的class中的成員money

```
class Player :public GameCharacter {  
    int money;  
  
    void Monster::fighting_system(Player* player) {  
        cout << "戰鬥結束" << endl;  
  
        int random_number = rand() % 30 + 10;  
        player->add_money(random_number);  
    }  
};
```

在戰鬥系統中，當怪物死亡時，會機生成數值，當作獲得的金錢。

```
bool Leader::trigger_event(Object* object) {  
    //give money to player  
    player->set_money(100); //initial money setting  
};
```

在Leader::trigger\_event()函式中設定角色初始金錢為100

# 7. 金錢系統

和商人購買物品時，會先偵測玩家身上的金錢是否充足，並依據結果決定交易是成功。

交易成功則會在玩家身上扣除購買該項物品所需的價錢。

```
bool Businessman::trigger_event(Object* object)
{
    if (player->get_money() < cost[choice - 1]) {    偵測玩家身上的金錢
        cout << "金錢不足，交易失敗" << endl << endl;
    }
    else {
        player->add_item(get_commodity()[choice - 1]);          扣除玩家身上的金錢
        player->set_money(player->get_money() - cost[choice - 1]);
        cout << "交易成功，花費 " << cost[choice - 1] << " 元，剩餘 " << player->get_money() << " 元。" << endl;
        if (get_commodity()[choice - 1]->get_tag() == "Weapon") {
            erase_item(choice - 1);
            cost.erase(cost.begin() + choice - 1);
        }
        cout << "" << endl;
    }
}
```

# Discussion

在這次作業中，花最多時間在地圖的設定上。為了隨機生成地圖，必多進行很多的偵測。除了要偵測房間本身的通道外，還要偵測地圖上的位置是否已經有房間存在。

一開始，我是利用寫好幾個 for 迴圈來偵測狀況，後來我嘗試利用陣列來記錄偵測的狀況，在Map 和 Room 中，確實讓程式簡單許多。這樣要隨機生成時，只需要隨機生成一定範圍的數字，就可以直接指向陣列中的特定元素，不用再額外的程式。

```
int Room::randomly_set_room(Room* r)
{
    vector<int> nums;
    int rand_number = -1;
    bool flag;
    for (int i = 0; i < 4; i++) {
        if (!test_path(i)) {
            nums.push_back(i);
        }
    }
}
```

```
vector<int> Map::check_near_room(Pos pos) {
    vector<int> restriction;
    //x方向
    if (pos.x == 0) {
        if (map[pos.y][pos.x + 1] != NULL) {
            restriction.push_back(1);
        }
    }
    else if (pos.x == map[0].size() - 1) {
        if (map[pos.y][pos.x - 1]) {
            restriction.push_back(0);
        }
    }
    else {
        if (map[pos.y][pos.x + 1] != NULL) {
            restriction.push_back(1);
        }
        if (map[pos.y][pos.x - 1]) {
            restriction.push_back(0);
        }
    }
    //y方向
    if (pos.y == 0) {
        if (map[pos.y + 1][pos.x] != NULL) {
            restriction.push_back(1);
        }
    }
    else if (pos.y == map.size() - 1) {
        if (map[pos.y - 1][pos.x]) {
            restriction.push_back(0);
        }
    }
    else {
        if (map[pos.y + 1][pos.x] != NULL) {
            restriction.push_back(1);
        }
        if (map[pos.y - 1][pos.x]) {
            restriction.push_back(0);
        }
    }
    return restriction;
}
```

# Conclusion

這次的 Dungeon 作業花費了許多時間，不過透過大量實作，也讓我更熟悉一些 OOP 的基本概念，如：繼承、virtual Function 等。在完成作業的過程中，我遇到許多複雜問題，像是偵測通道的方法，或是 trigger\_event 的設計。不過，在反覆的探索與嘗試中，最終也成功解決問題。未來在遇到相同或類似的問題，能更快的反應解法，或是找出更優秀的解法。