OOP HW1 - Dungeon

111550106 何義翔

1. 實作細節

Requirements

(1) 建圖、移動

我在 Room.h 中創建一個 struct 叫 RoomRecord，用來記錄每個房間的類別以及鄰居，並在 Dungeon::createMap() 中，創建這個 struct 的陣列用來存此地圖的資訊，並用迴圈一一創建各個房間再將其連結，如此一來若未來想修改地圖，只需修改此陣列的資訊即可。

移動部分，整個遊戲的主選單在 Dungeon::chooseAction() 中，當玩家選擇移動時呼叫 Dungeon::handleMovement()，並列出所有方向以及返回主選單五個選項，利用迴圈讓玩家選擇。若選擇的方向有路則直接呼叫 Player::changeRoom() 更新玩家的 currentRoom 及 previousRoom，並輸出房間編號以讓玩家更好掌握自己的位置；若無路則輸出錯誤資訊並讓玩家重新選擇；而若選擇返回則會回傳 False，不會與房間內的物件互動（Dungeon::handleEvent()）並直接結束 chooseAction()，如圖一。

```
if(action == "A" || action == "a"){
    if(handleMovement()){
        int size = player->getCurrentRoom()->getObjects().size();
        for(int i=0; i<size; i++){
            handleEvent(player->getCurrentRoom()->getObjects()[0]);
        }
    }
    return;
}
```

▲ 圖一（Dungeon.cpp: 257-265）

(2) 顯示角色資訊

當玩家選擇 Check Status，呼叫 player->triggerEvent()，此函式會輸出角色資訊（血量、金錢、武器等）。其中武器部分初始值為 NULL，因此僅需檢查 player 的 weapon 是否為 NULL 便可正常輸出。

(3) 戰鬥系統

程式中利用 vector<Object*> 儲存每個房間的物件。當玩家進到下個房間時，利用迴圈對該房間的每個物件執行 handleEvent()，當此物件是怪物時（用 getTag() 確認），詢問是否進行戰鬥，若否則回到上一個房間並回傳 false 終止迴圈及結束 chooseAction()；若欲進行戰鬥則呼叫 monster 的 triggerEvent()。

在 triggerEvent() 中，玩家與怪獸輪流攻擊，玩家在每次攻擊可選擇普攻、技能或大招，若使用技能或大招則呼叫 skill() 及 ultimate() 獲得相對應的效果（詳見下文職業部分），並設有 CD 機制，在玩家每次攻擊時更新 CD。而對於雙方的每次攻擊，都會輸出使用了何種攻擊方式、效果如何、被扣了多少血、雙方的狀態等。整個戰鬥系統為一迴圈，若玩家死亡則直接回傳 false，反之回傳 true。戰鬥結束後，若 triggerEvent() 回傳 true，則獲得金錢並將怪獸從房間中移除。

(4) 獲得道具、武器

欲拾起道具，先呼叫 Item 的 triggerEvent()，將傳入的 Object* 轉型再呼叫 player->addItem()。在 addItem() 中進行種類的判斷，若為道具則放進 inventory；若為武器則先判斷是否已裝備。重設數值，再根據武器是否專屬於玩家的職業，呼叫 Player::increaseStates() 更新能力值，並輸出裝備前後數值的變化，最後回到 addItem() 更新 player 的 weapon。

(5) NPC

遊戲共有兩個 NPC，分別為 Merchant 及 Mystery，當遇到他們時會觸發 NPC 的 triggerEvent()。Merchant，負責道具以及武器的買賣；Mystery 則是遊戲的小彩蛋，他會說一句類似亂碼的話，利用 base 64 解碼再用凱薩密碼還原便可得到 flag，將 flag 打在主選單便可獲得逆天武器 Voidreaper，增加 1000000 點攻擊力。

(6) 遊戲終止判斷

當玩家死亡（`player->checkIsDead() = True`）或打死 Boss（`player->getCurrentRoom()->getIsExit() = True`）則結束遊戲。

(7) 角色設計

另開檔案（`Occupation.h`）實作職業，每個職業皆繼承 `Player`，並將 `Player` 中的 `skill()` 及 `ultimate()` 設為 virtual function。每種職業的特點及技能如下：

i. Warrior（Shadow Warrior）：高血量、防禦力、CD 短
Skill（Empowered Strike）：強化攻擊（+50）
Ultimate（Demonic Decapitation）：怪獸血量低於 30% 直接斬殺

ii. Magician（Dark Sorcerer）：基礎能力低，但技能及大招效果極強
Skill（Last Embrace）：回 500 滴血（不影響 `maxHp`）
Ultimate(Death's Embrace)：15 倍攻擊力傷害

iii. Archer（Divine Archer）：普攻較痛、血量防禦低、CD 長
Skill（Celestial Piercer）：無視敵人防禦力
Ultimate（Divine Judgement）：隨機連續普攻 3 至 6 次

iv. 對應武器系統：在 `Item` 新增一個 member variable – `target`，紀錄對應的職業，若裝上不屬於自己的武器，則僅能獲得 40% 的能力值，同時也存在 `target = “Everyone”` 的道具，所有人都能正常使用。

(8) 貿易系統

在 `Object` 中新增 `int money`，對 `Player` 而言代表擁有的錢，對 `Monster` 則代表被殺死後會掉多少錢，`Item` 的 `money` 則標示購買所需的費用。當玩家遇到 Merchant 時會先列出所有商品，再詢問玩家是否要購買，這些商品包括強力武器及生命藥水，購買生命藥水將存於 `inventory` 而購買武器則直接更新玩家身上的武器。

Optional Enhancement

(9) 特殊輸入狀況（輸入錯誤）

為了避免使用者輸入錯誤或輸入與變數型態不合的資料，以增加程式的安全性，所有輸入的變數皆宣告為 string，並且輸入過程皆用迴圈包住，若輸入非預期的資料，則輸出錯誤訊息並請使用者重新輸入（如圖二），最後依需要轉成 int（如圖三）。

```
string direction;
while(true){
    cout << "> ";
    cin >> direction;
    Room* currentRoom = player->getCurrentRoom();
    if(direction == "A" || direction == "a"){···
    else if(direction == "B" || direction == "b"){···
    else if(direction == "C" || direction == "c"){···
    else if(direction == "D" || direction == "d"){···
    else if(direction == "E" || direction == "e"){···
    else cout << "[System] Invalid operation. Please choose from A to D.\n";
}
```

▲ 圖二（Dungeon.cpp: 139-179）

```
string schoice;
int ichoice;
while(true){
    cout << "> ";
    cin >> schoice;
    if(all_of(schoice.begin(), schoice.end(), ::isdigit)){
        ichoice = stoi(schoice);
    }
    else{
        cout << "[System] Invalid action. Please choose from 1 to " << commodity.size()+1 << ".\n";
        continue;
    }
```

▲ 圖三（NPC.cpp: 26-37）

(10) 繪製地圖

呼叫 Dungeon::printMap()，利用 '-'、'|'、'+' 繪製出輪廓，中間的地圖編號利用三元運算子判斷是否為當前房間，若是則輸出 "*"，否則直接輸出數字，如：((currentRoom == 8) ? "*" : "8")。

(11) 列出背包、使用道具

在主選單選擇打開背包時，會先呼叫 Player::listInventory()，此函式在玩家背包為空時僅輸出 empty 並回傳 false；若有道具則列出並回傳 true，再詢問是否使用。其中 listInventory() 會傳入一字串標示道具間的分隔，如此便能彈性讓道具分行印出或在同一行。（如圖四、五）

```
cout << "\nYour backpack: ";
if(!player->listInventory(" ")) return;
cout << "\n\nDo you want to use item? What do you want to use?\n";
player->listInventory("\n");
cout << player->getInventory().size() + 1 << ". No. Back to the last menu.\n";
```

▲ 圖四（Dungeon.cpp: 274-278）

```
void Player::useItem(Item* item){···

bool Player::listInventory(string space){
    if(inventory.empty()){
        cout << "<EMPTY>\n";
        return 0;
    }
    for(int i=0; i<inventory.size(); i++){
        cout << i+1 << ". " << inventory[i]->getName() << space;
    }
    return 1;
}
```

▲ 圖五（Player.cpp: 57-66）

(12) 存讀檔（Dungeon.cpp: 333-563）

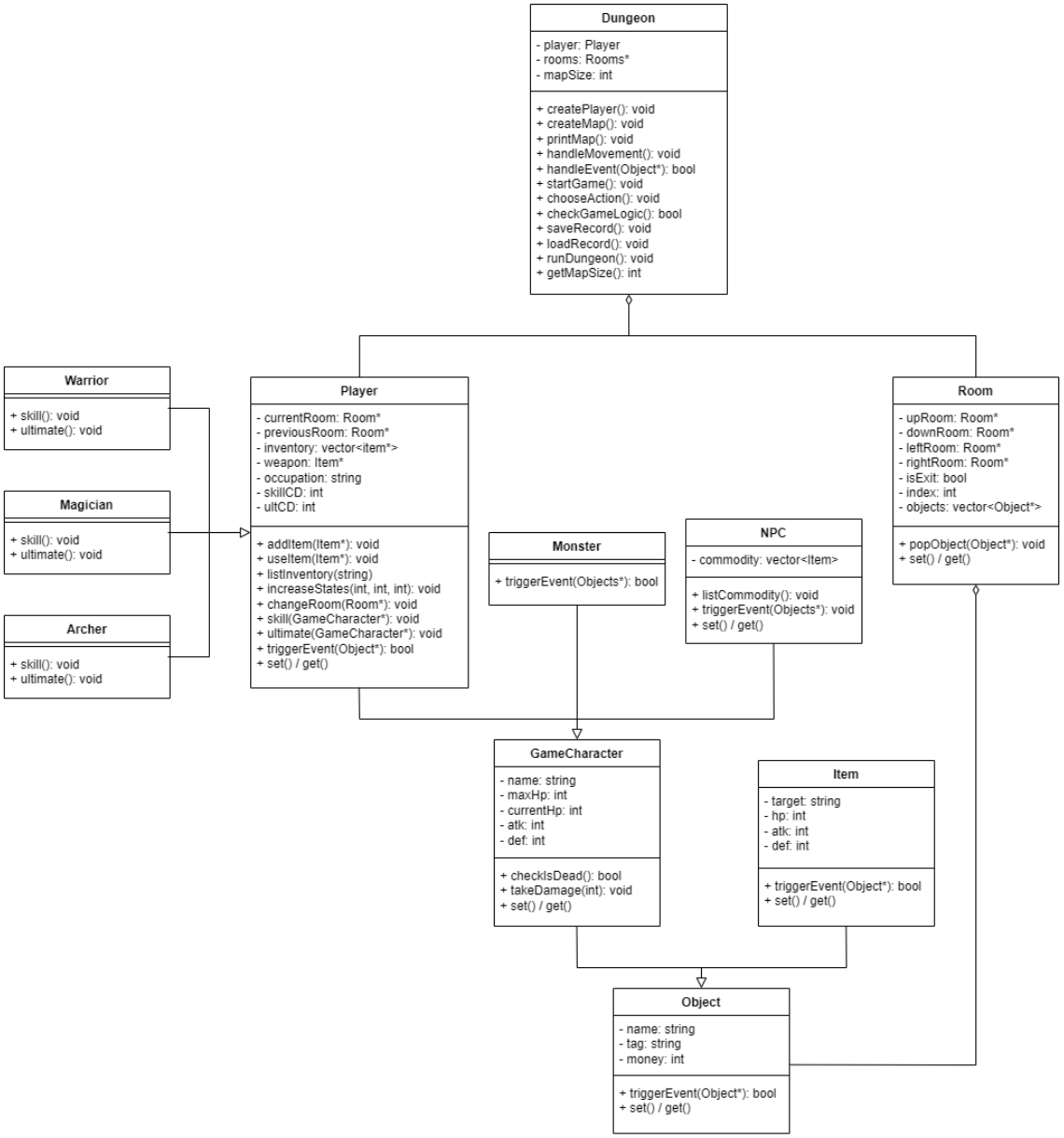
資料儲存

檔案利用 JSON 檔儲存，整個檔案為一個陣列，每個使用者的資訊為一個元素，使其能記錄不同使用者的資料。每個使用者當中的欄位包含玩家名稱 name，玩家資料 player，以及房間資訊 rooms，其中 player 中的 inventory、rooms 本身以及每個房間的物件皆用 JSON 的陣列儲存。

函式實作細節

首先引入 nlohmann/json.hpp（一個他人寫好的函式庫，可直接操作 JSON 檔。nlohmann/json: JSON for Modern C++ (github.com)），先利用 fstream 開啟檔案，判斷使用者資訊是否以在檔案中（先前存過檔），若無則創建一個 JSON 物件並存入使用者名稱、玩家的血量、當前房間、錢、職業、背包及每個房間所剩的物件，最後再加到 record.json。讀檔就跟創建玩家與地圖類似，根據資料創建。

2. UML


Dungeon
- player: Player
- rooms: Rooms*
- mapSize: int
+ createPlayer(): void
+ createMap(): void
+ printMap(): void
+ handleMovement(): void
+ handleEvent(Object*): bool
+ startGame(): void
+ chooseAction(): void
+ checkGameLogic(): bool
+ saveRecord(): void
+ loadRecord(): void
+ runDungeon(): void
+ getMapSize(): int
Warrior
+ skill(): void
+ ultimate(): void
Magician
+ skill(): void
+ ultimate(): void
Archer
+ skill(): void
+ ultimate(): void
Player
- currentRoom: Room*
- previousRoom: Room*
- inventory: vector<item*>
- weapon: Item*
- occupation: string
- skillCD: int
- ultCD: int
+ addItem(Item*): void
+ useItem(Item*): void
+ listInventory(string)
+ increaseStates(int, int, int): void
+ changeRoom(Room*): void
+ skill(GameCharacter*): void
+ ultimate(GameCharacter*): void
+ triggerEvent(Object*): bool
+ set() / get()
Monster
+ triggerEvent(Objects*): bool
NPC
- commodity: vector<Item>
+ listCommodity(): void
+ triggerEvent(Objects*): void
+ set() / get()
Room
- upRoom: Room*
- downRoom: Room*
- leftRoom: Room*
- rightRoom: Room*
- isExit: bool
- index: int
- objects: vector<Object*>
+ popObject(Object*): void
+ set() / get()
GameCharacter
- name: string
- maxHp: int
- currentHp: int
- atk: int
- def: int
+ checkIsDead(): bool
+ takeDamage(int): void
+ set() / get()
Item
- target: string
- hp: int
- atk: int
- def: int
+ triggerEvent(Object*): bool
+ set() / get()
Object
- name: string
- tag: string
- money: int
+ triggerEvent(Object*): bool
+ set() / get()

3. 運行成果

↓開始

```
What do you want to do?
1. Create a new game.
2. Load previous record.
> 1
 _____     ______     _____     __  __     ______
/\  __-.  /\  __ \   /\  __-.  /\ \/\ \   /\  ___\
\ \ \/\ \ \ \ \/\ \  \ \ \/\ \ \ \ \_\ \  \ \___  \
 \ \____-  \ \_____\  \ \____-  \ \_____\  \/\_____\
  \/____/   \/_____/   \/____/   \/_____/   \/_____/

[System] Welcome to COEUS, a world of darkness and despair.
[System] There, the only law is strength and wisdom.
[System] Will you rise to the top, or be swallowed by the abyss? It's all up to you.
[System] Now let's start. Bless you.

Type in your name: Sean

Choose your occupation:
A. Shadow Warrior (with much HP and Def)
B. Dark Sorcerer (with powerful skill and ultimate)
C. Divine Archer (with high normal attack)
> A
```

↓主選單 + 移動

```
What do you want to do?
A. Move
B. Check Status
C. Open Backpack
D. Check Map
E. Save Record
> a

Where do you want to go?
A. Go up
B. Go down
C. Go left
D. Go right
E. Back to menu
> a
```

↓戰鬥

```
[System] You start a fight with Monster

Sean: 1500/1500
Monster: 600/600

What do you want to use
A. Normal attack
B. Skill (0/1)
C. Ultimate (0/3)
> c

You use Demonic Decapitation!
Monster take 175 points of demage.

Sean: 1500/1500
Monster: 525/600

You take 50 points of demage.

Sean: 1450/1500
Monster: 525/600

What do you want to use
A. Normal attack
B. Skill (0/1)
C. Ultimate (3/3)
> b

You use Empowered Strike!
Monster take 125 points of demage.
And Monster was dizzy. You can continue attack.
```

↓繪製地圖

```
           +-----+
           |  8  |
           +-----+
              |
+-----+     +-----+
|  6  |     |  7  |
+-----+     +-----+
   |           |
+-----+     +-----+     +-----+
|  3  |-----|  *  |-----|  4  |
+-----+     +-----+     +-----+
              |
           +-----+     +-----+     +-----+
           |  0  |-----|  2  |-----|  5  |
           +-----+     +-----+     +-----+
```

↓玩家資訊

```
[Sean's Status]
> HP: 1500/1500
> ATK: 130
> DEF: 50
> MONEY: 20
> WEAPON: Wand
```

↓戰鬥結束 + 拾起武器（非專武）

```
[System] You killed Monster!
[System] You earn 20 dallars from Monster.

You find a weapon, Wand. Do you want to pick up? (Y/N): y
You equipped with Wand.
This weapon is not for you. You can just abtain some of its ability.
[HP] 1350/1500 -> 1350/1500
[ATK] 100 -> 130
[DEF] 50 -> 50
```

↓遇到 Merchant + 買武器（專武）

```
[Room 4]

[System] You meet a merchant.
[Merchant] What do you want to buy, Sean?
[Merchant] Notice: you can just buy one item.
1. Abyssal Sword $150    2. Void Staff $150    3. Starfall Arrow $150    4. Spirit Visage $150    5. Health Potion $100    6. I don't want anything.
> 1
You equipped with Abyssal Sword.
This weapon is specially for you.
[HP] 1150/1500 -> 1150/1500
[ATK] 130 -> 300
[DEF] 50 -> 50
```

↓列出背包 + 使用道具

```
What do you want to do?
A. Move
B. Check Status
C. Open Backpack
D. Check Map
E. Save Record
> c

Your backpack: 1. Health Potion

Do you want to use item? What do you want to use?
1. Health Potion
2. No. Back to the last menu.
> 1
[HP]: 1300/1500 -> 1500/1500
```

4. 討論

(1) 實作遇到的困難

整個程式的邏輯

我覺得這是最困難的部分，要一直追蹤程式運行的進度，程式會如何呼叫函式、需不需要回傳等等。我自己在面對這類型的 bug 是多寫幾個測試的輸出，看看程式有沒有依我所想的順序進行，或慢慢縮小問題範圍。

太多分支

整個遊戲有很多選擇，尤其我想處理輸入錯誤的問題，導致要讓每個分支都能回到同樣的遊戲進行（`chooseAction()`）變得十分困難，像是開發過程中常常面對這個情況最後要 `break`、`return`、`continue`，還是呼叫另一個函式之類的問題，還有輸出給使用者的訊息要寫在哪裡才不會多輸出或少輸出。對於這問題我只能不斷執行程式，確認還有哪些順序上的問題要修。

創建房間

在一開始創建房間時，我是對每個房間一個一個設定其鄰居，但剛開始我一直改地圖，導致每次更改都要花很多時間更新，後來我直接設計一個 struct 紀錄每個房間的資訊，就能用迴圈處理了。

存檔功能

若僅用原先的 `ftream`，只能將變數值存入檔案中（要存變數也可以啦，只是在讀檔的時候還要額外處理），導致檔案基本上不可讀，而且存檔和讀檔的 code 也很難讀懂，因此我才想用 JSON 檔，讓檔案較有系統。

但首先就面臨到，如何用 C++ 處理 JSON 檔？還好已有現成的工具可使用，但接著就是學習該函式庫如何使用，這也花了我不少時間（主要是在找我想要的功能是否存在），再來就是 `record.json` 該如何設計，最後我想實作多個使用者的存檔，就直接用一個大陣列存。

(2) 當前程式的問題

程式架構

我覺得現階段 Dungeon 程式架構仍偏亂，風格沒有統一性，有些函式也為了實作方便，儘管他是對一個 class 的操作，但卻宣告在 `Dungeon.h` 中。

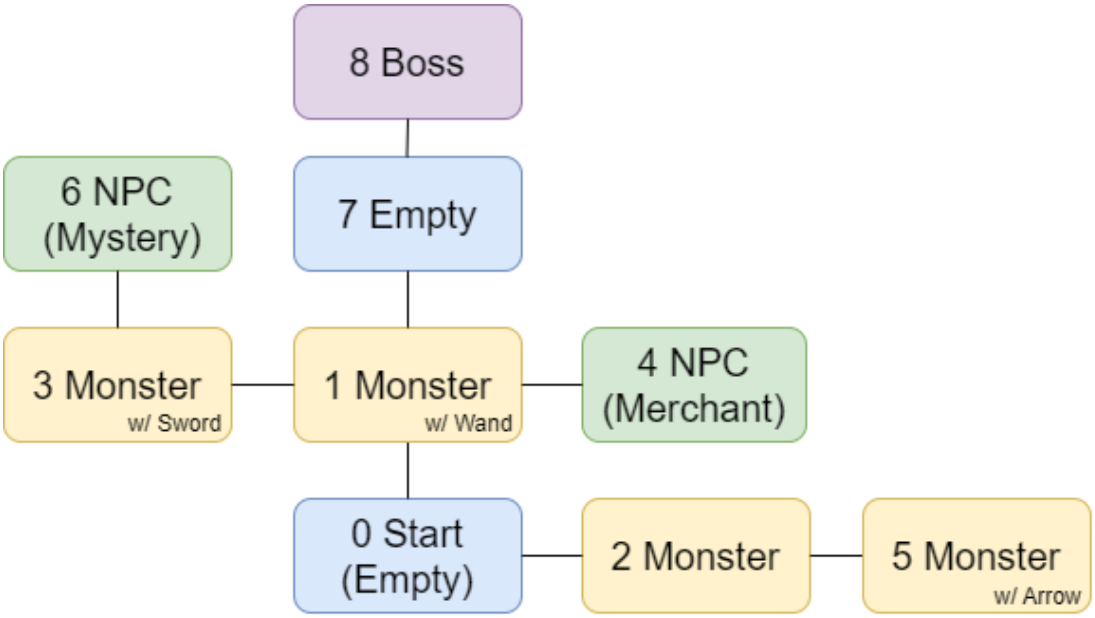
函式功能不完全

有些功能（如拾起道具）並不完全在該函式（`Item::triggerEvent()`）中，或甚至完全在另一個函式（`addItem()`），這導致程式雖然可以運行，但可讀性低，debug 的過程也需要不斷在不同檔案、不同函式間切換。

5. 結論

這個 Dungeon 用了許多 OOP 的概念，如繼承、virtual function 等，或甚至是 JSON 檔本身都有 OOP 的概念，雖然過程遇到了很多問題，但都讓我對 OOP，或說對 C++ 這個語言有更深入的了解。然而，我還有許多功能想實作如讓使用者能輸入密碼並對密碼進行加密，讓其更像一般遊戲的登入，或者新增開關門機制，讓玩家需獲得特定道具才能前進，或關門讓玩家無法後退等等。前者我已經找到相對應的函式庫了，但礙於時間關係尚未去讀其文件，而後者我想多增加一些 `Item` 就能解決了。

6. 附錄：地圖


8 Boss
6 NPC
(Mystery)
7 Empty
3 Monster
w/ Sword
1 Monster
w/ Wand
4 NPC
(Merchant)
0 Start
(Empty)
2 Monster
5 Monster
w/ Arrow