Dungeon Report

109550059 Yen-chieh Huang[*]

Contents

[*]I cannot type in my Chinese name in LaTeX :(

1 Implementation Detail

1.1 Class Hierarchy

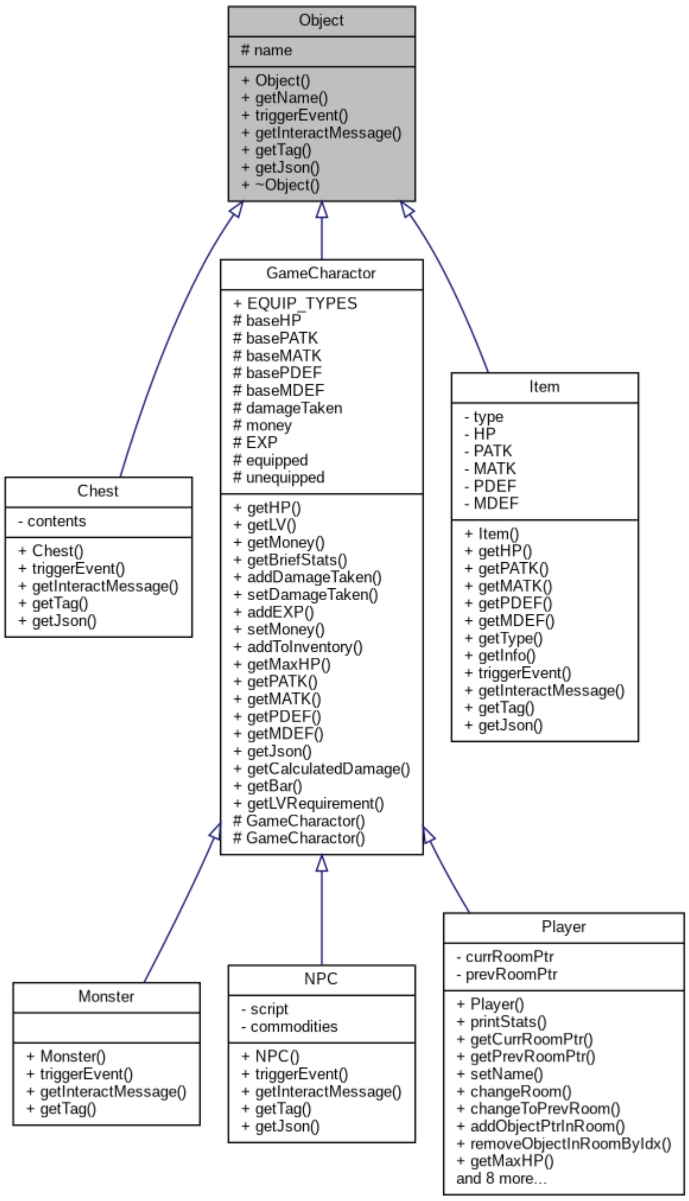


Figure 1: Object's UML Diagram

1.2 Actions Menu

The actions menu is implemented by checking whether the current room is connected to other rooms, and then loop through all objects in the room (by calling their virtual function `getInteractMessage()`). If there are monsters in the room, then all rooms except the previous one are locked. In this case, the only movement player can make is retreat.

Each action is assigned with a special character (e.g. *a* represents moving to the left room, *1* might be talking to NPC).

Once the program prints out all available actions, it then prompts user to input a character. If user input *w*, *a*, *s*, or *d*, the player will move to its corresponding room. If user input *e*, player's stats and inventory will be printed out. If the input is a numeric value (which signifies objects' unique number in that room), that object's virtual function `triggerEvent()` will be called.

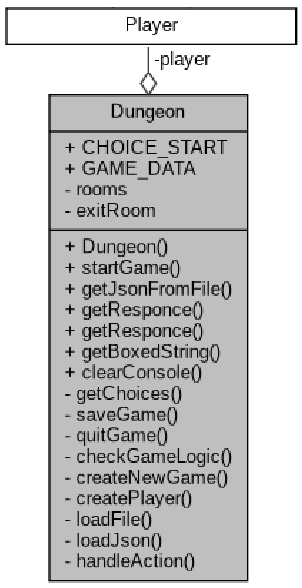


Figure 2: Dungeon's UML Diagram

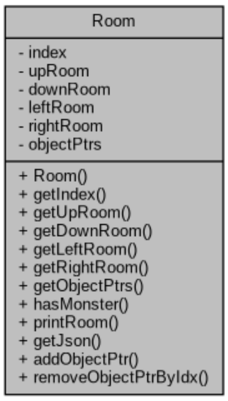


Figure 3: Room's UML Diagram

1.3 Movement

The movement is simply done by calling player's private method `changeRoom()`. It can update `Player`'s `currRoomPtr` and `prevRoomPtr`, which are pointer to current room and pointer to previous room, respectively.

`Player`'s private method `changeToPrevRoom()` can move to previous room. It is used when player retreats.

1.4 Showing Status

Every GameCharacter has 9 properties: LV, EXP, HP, $PATK$, $MATK$, $PDEF$, $MDEF$[1], $Money$, and $Inventory$ (including equipped items and unequipped items).

LV is calculated by $\left\lfloor \sqrt{\frac{EXP}{30}} \right\rfloor + 1$.

HP is calculated by $baseHP * M + E_1$;

PATK is calculated by $basePATK * M + E_2$;

PDEF is calculated by $basePDEF * M + E_3$, where

$$M = \begin{cases} 1 + 0.05(LV-1)^2, & \text{if it is a player} \\ 1, & \text{otherwise} \end{cases}$$

$$E_1 = \text{sum of } HP \text{ of all equipped items}$$
$$E_2 = \text{sum of } PATK \text{ of all equipped items}$$
$$E_3 = \text{sum of } PDEF \text{ of all equipped items}$$

$MATK$ and $MDEF$ is calculated in a similar way.

`Player`'s private method `printStats()` can print out his/her status in a human-readable format.

1.5 Pick up Items

There are 7 item types in this game - *Helmet*, *Chestplate*, *Leggings*, *Boots*, *Shield*, *Weapon*, and *Usable*. *Usable* can only be used one time, yet the others can be equipped and increase `GameCharacter`'s *HP*, *PATK*, *MATK*, *PDEF* or *MDEF*.

Player can pick up and equip *Helmet*, *Chestplate*, *Leggings*, *Boots*, *Shield* and *Weapon* automatically whenever the corresponding slot is empty. If the slot is not empty, then the item to be picked up will go into Player's "not equipped" slot.

[1] "P" means "physical", whereas "M" means "magical". For example, "PATK" represents "physical attack".

Player can change their equipment by entering e in actions menu, which calls `Player`'s virtual method `triggerEvent()`.

1.6 Fighting System

Player can attack monsters by selecting "Monster: attack [*monster's name*]" in actions menu, which calls `Monster`'s virtual function `triggerEvent()`. The damage in the fighting system is calculated by

$$round(PATK * \frac{1}{1+\frac{PDEF}{100}} * r_1 + MATK * \frac{1}{1+\frac{MDEF}{100}} * r_2)$$

where $PATK/MATK$ is attacker's $PATK/MATK$, $PDEF/MDEF$ is defender's $PDEF/MDEF$, r_1 and r_2 is a random number in normal distribution with $\mu = 1$ and $\sigma = 0.1$.

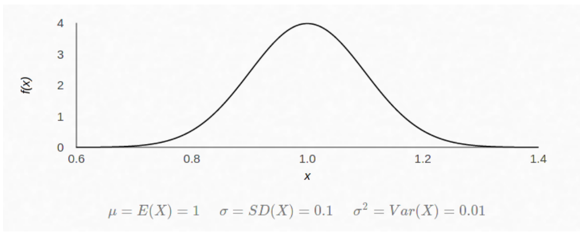


Figure 4: Normal distribution with $\mu = 1$ and $\sigma = 0.1$.

1.7 NPC

Every NPC has his own `script` and a vector of `commodities`. Player can talk to NPC by selecting "NPC: Talk to [*NPC's name*]" in actions menu, which calls NPC's virtual function `triggerEvent()`.

When `NPC`'s `triggerEvent()` is called, `NPC`'s `script` and a list of `commodities` will print out. `Player` can select the item he/she want to buy.

Every commodity in `commodities` has the type `pair<int, Item>`, where `int` is the price of the item, and `Item` is the item to be sold. If `Player`'s `money` is larger than the price of the commodity, then he/she can buy the item. Otherwise, "You don't have enough money!" will print out.

1.8 Game Logic

In this project, game state is defined as a enum with three possible values: *win*, *lose*, or *indeterminate*. When player is in the exit room, then the game state is *win*. If player's $HP = 0$, then the game state is *lose*. Otherwise, it is *indeterminate*.

1.9 Record System

The record system is implemented by saving/loading json file on disk. In fact, all game data can be represented as a `nlohmann::json`[2] object. The json file is highly human-readable and structurized, so the game can be customized easily. The map in this game can be changed by modifying `GAME_DATA` in `Dungeon.cpp`, or simply editing the saved json file.

Every class excluding `Dungeon` has its own `getJson()` method, which can return a `nlohmann::json` object. Game data is saved by subsequently calling `getJson()` of every class. Similarly, Every class excluding `Dungeon` has its own constructor that accepts `nlohmann::json` object as its parameter. Game data is loaded by subsequently calling constructor of every class.

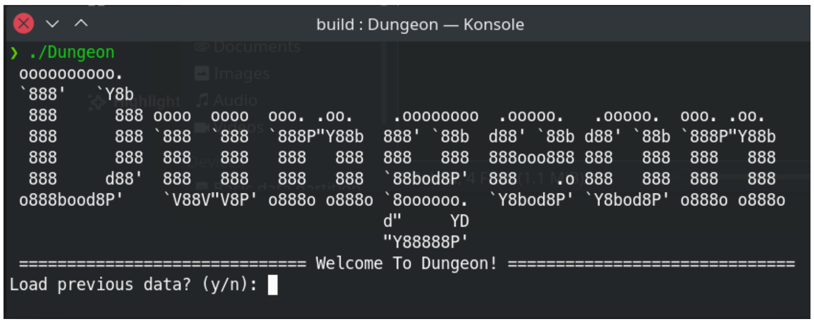
1.10 Optional Enhancement

- LV, EXP System
  - I added LV(level) and EXP(experience) system in this game. Player's health, attack and defense not only depends on his/her equipment, but also depends on his/her level. (See 1.4 for further detail.)

[2]https://github.com/nlohmann/json

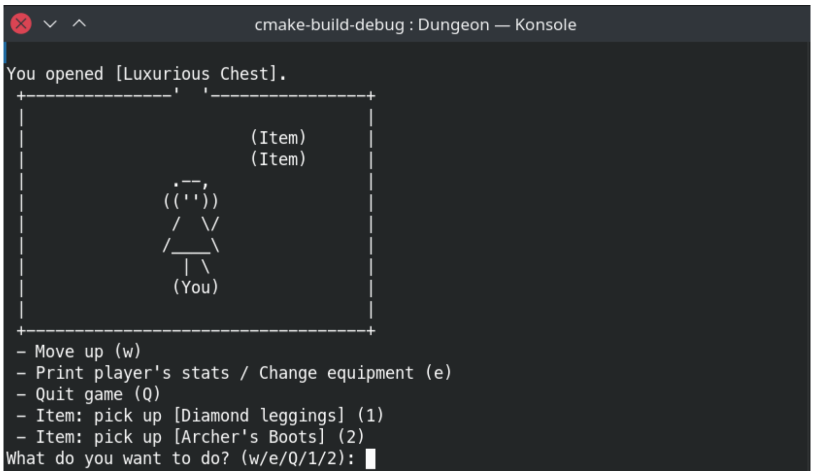
- Enhanced Fighting System
  - ATK/DEF in this game are divided into two types: physical and magical. Different types of ATK can deal different damage to monsters or player. Similarily, different types of DEF can shield different types of ATK. (See 1.6 for further detail.)
- Skill
  - Player unlocks skills when they reaches some certain levels. Every skill has its own properties, including $PATK$, $MATK$, HP and CD. CD stands for "cooldown". Player can use a skill only if that skill is ready (when CD equals to 0). Each attack can reduce CD by one.
- Equipment System
  - Player can choose a variety of weapons and armors to prepare for different combat situation. For example, armors that can shield magical damage are good choices when attacking *wizard* or *magician*; weapons that has high physical damage are excellent choices when attacking monsters that are weak in physical defense.
  - Furthermore, I divided items in this game into 7 categories: *Helmet*, *Chestplate*, *Leggings*, *Boots*, *Shield*, *Weapon* and *Usable*. Player can only equip one equipment of the same type (*Usable* can only be used once).
- Drop Items When Monster Dies
  - When monster dies, its equipment will drop onto the ground. That monster's EXP and $Money$ will also transfer to player. If player's LV upgrades after defeating the monster, his/her HP will be refilled.
- Trading System
  - I added money in this game, player can use money to trade with NPC. (See 1.7 for further detail.)
- Customizable Game
  - Game data is written in json file, so the game is highly customizable. (See 1.9 for further detail.)
- New Object: Chest
  - I added chest in this game. When player opens a chest, then items in that chest can be picked up.
- Use Smart Pointer to Prevent Memory Leak
  - I used `shared_ptr` extensively in this project. This smart pointer can avoid the problem of dangling pointer, so memory leak would not occur.
- Better Gaming Experience
  - I've made a lot of optimization on the interface of the game to make the game more attractive. (See section 2 for further detail.)
- This Report XD
  - I've made great effort on this report. To my mind, being able to describe how this game works is just as important as coding it.

2 Results



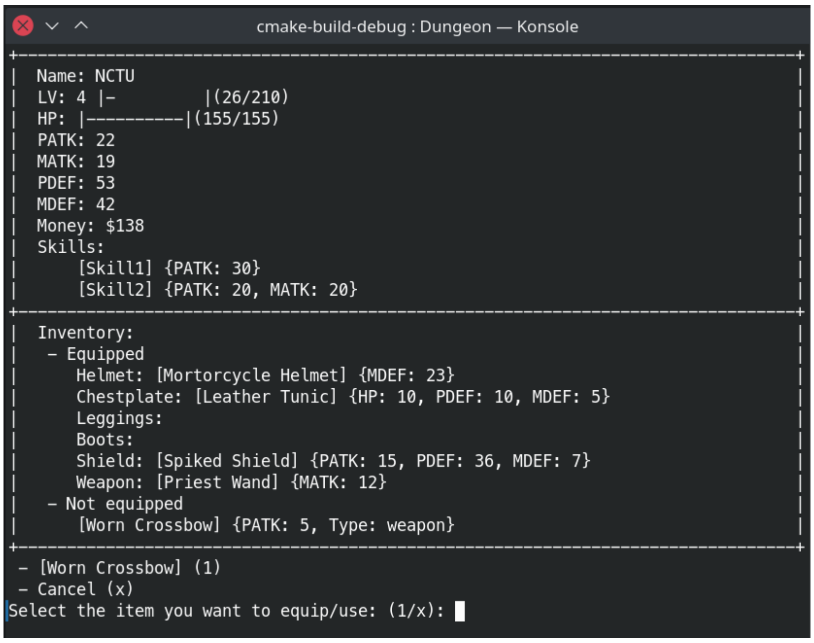
```
❯ ./Dungeon
 oooooooooo.
 `888'   `Y8b
  888      888 oooo  oooo  ooo. .oo.    .oooooooo  .ooooo.   .ooooo.  ooo. .oo.
  888      888 `888  `888  `888P"Y88b  888' `88b  d88' `88b d88' `88b `888P"Y88b
  888      888  888   888   888   888  888   888  888ooo888 888   888  888   888
  888     d88'  888   888   888   888  `88bod8P'  888    .o 888   888  888   888
 o888bood8P'    `V88V"V8P' o888o o888o `8oooooo.  `Y8bod8P' `Y8bod8P' o888o o888o
                                       d"     YD
                                       "Y88888P'
 ============================== Welcome To Dungeon! ==============================
Load previous data? (y/n): █
```

Figure 5: welcome screen



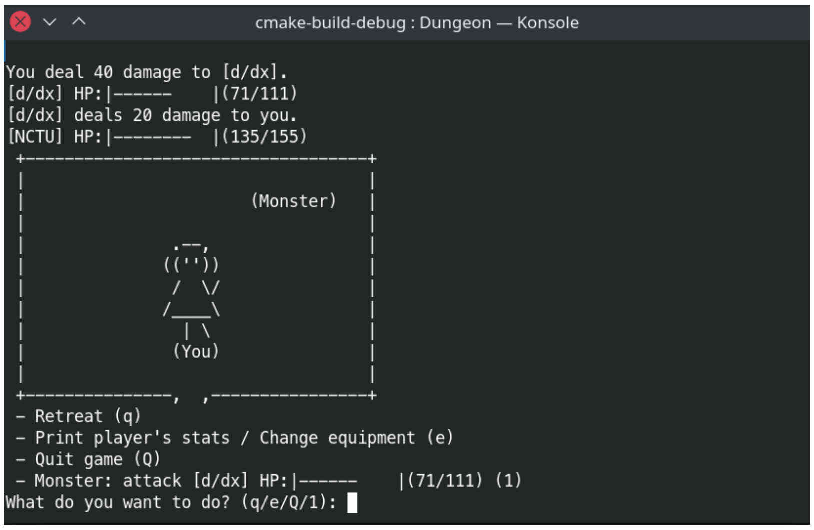
```
You opened [Luxurious Chest].
 +----------------'  '----------------+
 |                                    |
 |                        (Item)      |
 |                        (Item)      |
 |                .--,                |
 |               ((''))               |
 |                /  \/               |
 |               /____\               |
 |                 | \                |
 |                (You)               |
 |                                    |
 +------------------------------------+
 - Move up (w)
 - Print player's stats / Change equipment (e)
 - Quit game (Q)
 - Item: pick up [Diamond leggings] (1)
 - Item: pick up [Archer's Boots] (2)
What do you want to do? (w/e/Q/1/2): █
```

Figure 6: actions menu / open chest / pick up items



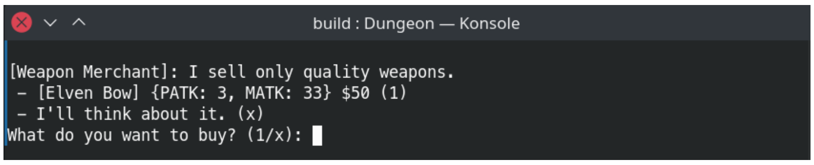
```
+--------------------------------------------------------------------------------+
|  Name: NCTU                                                                    |
|  LV: 4 |-         |(26/210)                                                    |
|  HP: |----------|(155/155)                                                    |
|  PATK: 22                                                                      |
|  MATK: 19                                                                      |
|  PDEF: 53                                                                      |
|  MDEF: 42                                                                      |
|  Money: $138                                                                   |
|  Skills:                                                                       |
|      [Skill1] {PATK: 30}                                                       |
|      [Skill2] {PATK: 20, MATK: 20}                                             |
+--------------------------------------------------------------------------------+
|  Inventory:                                                                    |
|   - Equipped                                                                   |
|      Helmet: [Mortorcycle Helmet] {MDEF: 23}                                   |
|      Chestplate: [Leather Tunic] {HP: 10, PDEF: 10, MDEF: 5}                   |
|      Leggings:                                                                 |
|      Boots:                                                                    |
|      Shield: [Spiked Shield] {PATK: 15, PDEF: 36, MDEF: 7}                     |
|      Weapon: [Priest Wand] {MATK: 12}                                          |
|   - Not equipped                                                               |
|      [Worn Crossbow] {PATK: 5, Type: weapon}                                   |
+--------------------------------------------------------------------------------+
 - [Worn Crossbow] (1)
 - Cancel (x)
Select the item you want to equip/use: (1/x): █
```

Figure 7: print player's stats / change equipment



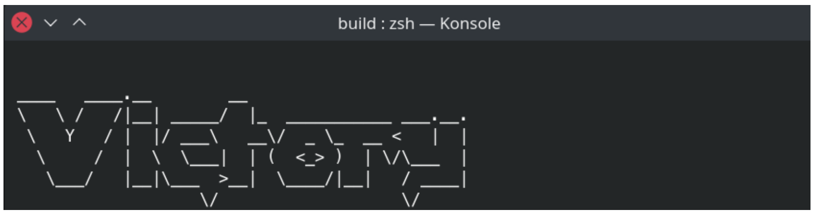
```
You deal 40 damage to [d/dx].
[d/dx] HP:|------    |(71/111)
[d/dx] deals 20 damage to you.
[NCTU] HP:|--------  |(135/155)
 +------------------------------------+
 |                                    |
 |                       (Monster)    |
 |                                    |
 |               .--,                 |
 |              ((''))                |
 |               /  \/                |
 |              /____\                |
 |                | \                 |
 |               (You)                |
 |                                    |
 +---------------,  ,-----------------+
 - Retreat (q)
 - Print player's stats / Change equipment (e)
 - Quit game (Q)
 - Monster: attack [d/dx] HP:|------    |(71/111) (1)
What do you want to do? (q/e/Q/1):
```

Figure 8: fight with monster



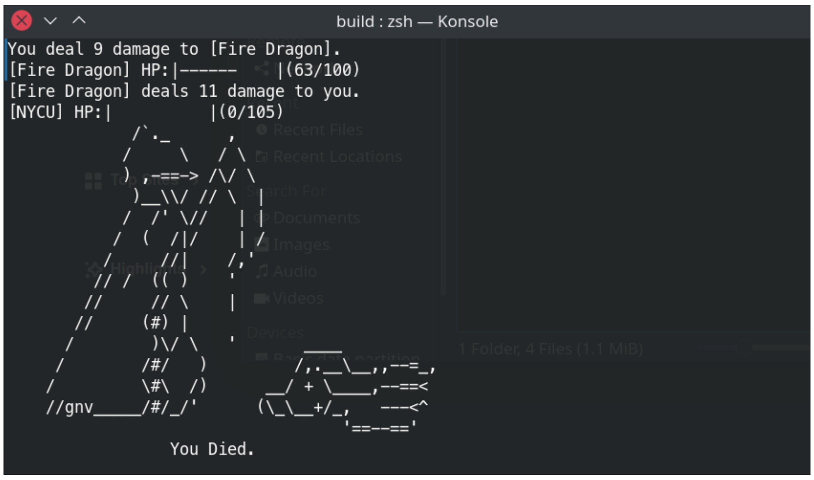
```
[Weapon Merchant]: I sell only quality weapons.
 - [Elven Bow] {PATK: 3, MATK: 33} $50 (1)
 - I'll think about it. (x)
What do you want to buy? (1/x):
```

Figure 9: talk to NPC



```
 ____   ____.__        __
 \   \ /   /|__| _____/  |_  ___________ ___.__.
  \   Y   / |  |/ ___\   __\/  _ \_  __ <   |  |
   \     /  |  \  \___|  | (  <_> )  | \/\___  |
    \___/   |__|\___  >__|  \____/|__|   / ____|
                    \/                   \/
```

Figure 10: win



```
You deal 9 damage to [Fire Dragon].
[Fire Dragon] HP:|------    |(63/100)
[Fire Dragon] deals 11 damage to you.
[NYCU] HP:|          |(0/105)
             /`._      ,
            /     \   / \
            ) ,-==-> /\/ \
             )__\\/ // \  |
            /  /' \//   | |
           /  (  /|/    | /
          /     //|    /,'
         // /  (( )    '
        //     // \    |
       //     (#) |
      /        )\/ \   '      ____
     /        /#/   )         /,.__\__,,--=_,
    /         \#\  /)      __/ + \____,--==<
    //gnv_____/#/_/'      (\_\__+/_,   ---<^
                                   '==--=='

                 You Died.
```

Figure 11: lose

3 Discussion

In this section, I'll talk about some bad designs I've made in this project. Although these bad designs don't affect the functionality of the program, it can have great impact on extensibility and readability of the project. Therefore, I think it's necessary to record these mistakes, so that I can avoid these errors in the future.

3.1 The parameters in `triggerEvent()` should be modified

In this project, the function signature of `triggerEvent` is `triggerEvent(Object&)`. However, it seems that `triggerEvent(Player&)` would make more sense, since only `Player` can interact with `Object`. If `triggerEvent` accepts `Object&` as its parameter, then it means that `Object` can interact with other `Object`, but I've not implemented this feature so far.

Therefore, I think it's better to substitute `Object&` with `Player&` in `triggerEvent()`. However, it can lead to another problem - **circular dependency** - because `triggerEvent(Player&)` is declared in `Object`, `Player` is also inherited from `Object`. Compile errors rises after I changed `Object&` to `Player&` in `triggerEvent()`'s parameter list.

3.2 Interactable objects should be distinguished from ordinary `Object`

In this game, `Room` has a member variable `objectPtrs`, which has the type `vector<shared_ptr<Object>>` and it can store pointers of objects in that room. However, it's unreasonable to store `Player*` in `objectPtrs`, since there are only one `Player` in the game. However, it is allowed to do so, because `Player` is an `Object`. When a `Player*` is stored in `objectPtrs`, what should the program do when interacting with that `Player`? It's kind of weird to interact with other `Player` in this single-player game.

Therefore, I think we should distinguish ordinary `Object` from interactable objects, i.e. `Chest`, `Item`, `Monster`, and `NPC`. Maybe we can create a new class named `InteractableObject`, and place `triggerEvent()` in it. `Chest`, `Item`, `Monster` and `NPC` are inherited from both `Object` and `InteractableObject` (by using multiple inheritance). In this way, the data type of `objectPtrs` can be changed to `vector<shared_ptr<InteractableObject>>`, so `Player` would not be considered as an interactable object (because it is inherited only from `Object`).

4 Conclusion

This project was made possible thanks to various advantages of OOP principles. OOP allowed this project to become more concise, well-organized, and easy-to-read. On working with this project, I encounter many difficulties related to OOP. By tracing errors generated at compilation times, I learnt a lot when solving these problems. I'm sure that I become much more familiar with C++ after finishing this project.