OOP Homework 1 - Dungeon

Hsuan-Yu Kuo

May 1, 2021

Contents

1 Introduction

This is the Dungeon project for the NYCU OOP & DS lecture. Other than the specification described, my optional enhancements are as follows:

1. In-game currency: Player can possess some money in his/her hand to buy items. Furthermore, player can gain money by opening chests or killing monsters.
2. Randomized algorithm for generating the dungeon.
3. Beautified TUI.

All the source codes are in the zip file and on my GitHub page.

2 Implementation Details

2.1 Game Introduction

According to the specification, this is a text-based RPG with the player exploring a dungeon and fight against unknown monsters.

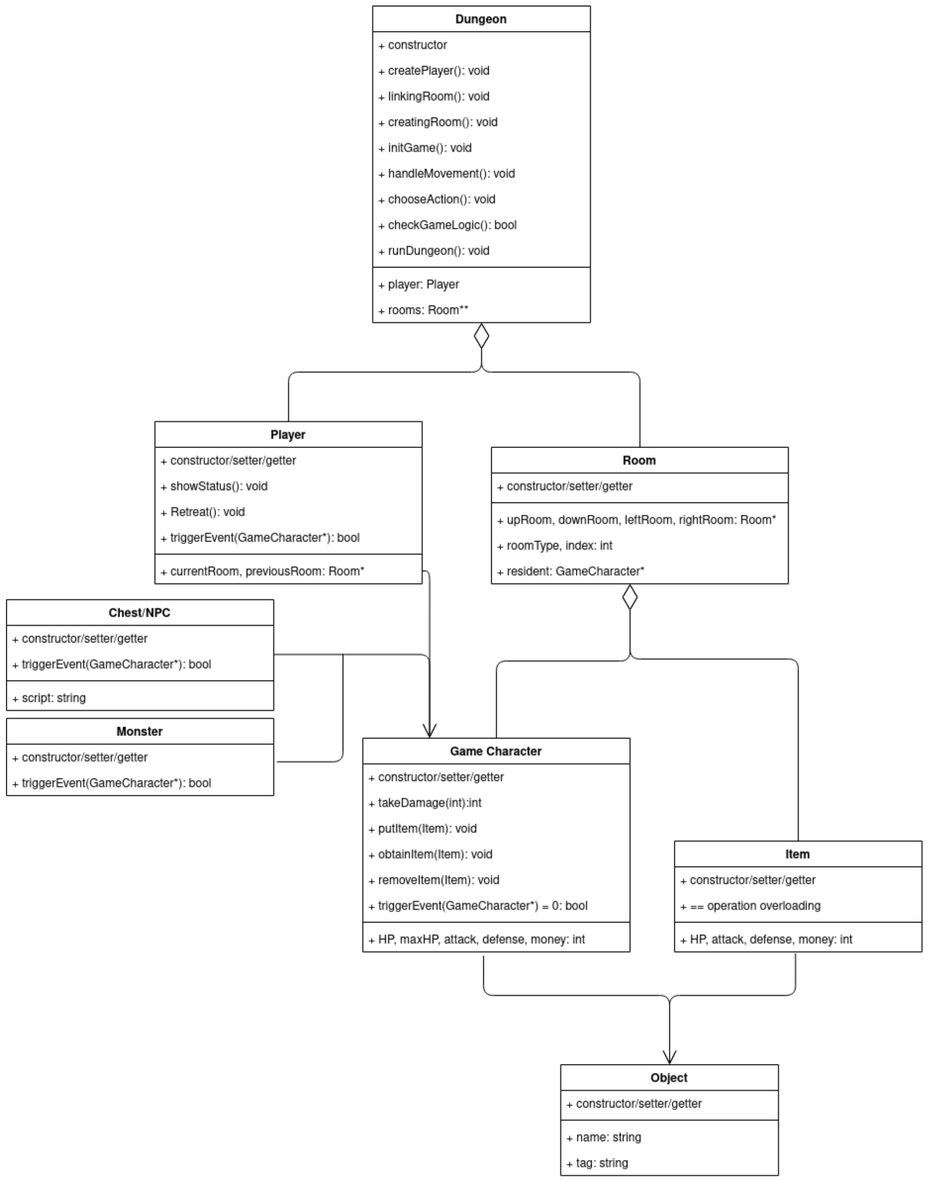

Dungeon
+ constructor
+ createPlayer(): void
+ linkingRoom(): void
+ creatingRoom(): void
+ initGame(): void
+ handleMovement(): void
+ chooseAction(): void
+ checkGameLogic(): bool
+ runDungeon(): void
+ player: Player
+ rooms: Room**
Player
+ constructor/setter/getter
+ showStatus(): void
+ Retreat(): void
+ triggerEvent(GameCharacter*): bool
+ currentRoom, previousRoom: Room*
Room
+ constructor/setter/getter
+ upRoom, downRoom, leftRoom, rightRoom: Room*
+ roomType, index: int
+ resident: GameCharacter*
Chest/NPC
+ constructor/setter/getter
+ triggerEvent(GameCharacter*): bool
+ script: string
Monster
+ constructor/setter/getter
+ triggerEvent(GameCharacter*): bool
Game Character
+ constructor/setter/getter
+ takeDamage(int):int
+ putItem(Item): void
+ obtainItem(Item): void
+ removeItem(Item): void
+ triggerEvent(GameCharacter*) = 0: bool
+ HP, maxHP, attack, defense, money: int
Item
+ constructor/setter/getter
+ == operation overloading
+ HP, attack, defense, money: int
Object
+ constructor/setter/getter
+ name: string
+ tag: string

Figure 1: UML of the project.

2.2 Object Class

The base class for Item and Game Character Class.

Objects implemented:

- **name**: Object name
- **tag**: Object category

Functions and other members implemented:

- Constructors, Setters & Getters

2.3 Game Character Class

The base class for Chest, Player, NPC, Monster . Inherited from Object.

Objects implemented:

- **HP, maxHP, attack, defense, money**: Character features.
- A vector of Item that stores the object the character possesses.

Functions and other members implemented:

- Constructors, Setters & Getters
- **int takeDamage(int AnAttack)**: The character take $\max(0, \text{AnAttack} - \text{defense})$ damage, and return how many damage the character take actually.
- **int putItem(Item obtain)**: The character take the effects of the item, e.g., gain money, gain HP, reduce attack. Then the item will be put in the vector of Item.
- **int obtainItem(Item obtain)**: Almost same as **putItem**, but don't take the effects of the item.
- **int removeItem(Item obtain)**: Remove the item *obtain* from the vector. Uses the **count()** and **remove()** function in **algorithm** and **vector** library respectively.
- **virtual bool triggerEvent(GameCharacter* player) = 0**: A pure virtual function to implement what to do in each room.

2.4 Item Class

The class for items gained by characters. Inherited from Object.

Objects implemented:

- **HP, attack, defense, price**: Item features, refer to gain HP, gain attack, gain defense, reduce or gain money, respectively.

Functions and other members implemented:

- Constructors, Setters & Getters
- Overloaded == operator to let the **count()** function in **removeItem()** run smoothly.

2.5 Player Class

The class for game player (user). Inherited from Game Character class.

Objects implemented:

- **currentRoom, previousRoom**

Functions and other members implemented:

- Constructors, Setters & Getters
- **showStatus()**: Dump out the player's information.
- **Retreat()**: Retreat if needed when fighting monsters.
- **virtual bool triggerEvent(GameCharacter* player)**: Nothing is implemented here, just a function to satisfy the requirements of virtual functions.

2.6 Monster Class

The class for monsters. Default monsters are Slime and Boss. Inherited from Game Character class.

Functions and other members implemented:

- Constructors, Setters & Getters
- **virtual bool triggerEvent(GameCharacter* player)**: Implements the fighting system. Players can choose to fight the monster or retreat, and then the monster will fight back if the player chooses to attack. The process loops until the player retreats or one of them have non-positive HP.

2.7 Chest Class

The class for room of chests. There should be no character in chest rooms, but I've set a virtual "character" to treat the chest room the same as other room types. Default chests are sword (add attack), snake (reduce attack), money (gain money). Inherited from Game Character class.

Objects implemented:

- **script**: The script shown when user wants to open a chest.

Functions and other members implemented:

- Constructors, Setters & Getters
- **virtual bool triggerEvent(GameCharacter* player)**: Implements picking up a chest. The chest list is stored in the vector of item (in Game Character class). When user wants to pick up a chest, the order of the chests are random shuffled, and the name of the items are hidden to enhance game experience. The player then will be equipped with the item (the item will be stored in the vector of the player.)

2.8 NPC Class

The class for the shop. Very similar to Chest class except the names of the items are not hidden. Inherited from Game Character class.

Objects implemented:

- **script**: The script shown when user wants to buy something.

Functions and other members implemented:

- Constructors, Setters & Getters
- **virtual bool triggerEvent(GameCharacter* player)**: Implements talking to Shop and buy items. Default inventories are sword (add attack), shield (add defense), fruit (gain HP). Player should have enough money to buy the inventories and get item from NPC. I assume Shop have infinite supply of the items, so the items would not disappear.

2.9 Room Class

The class for Rooms.

Objects implemented:

- **upRoom, downRoom, leftRoom, rightRoom**: Pointers of adjacent rooms.
- **GameCharacter* resident, int roomType, int index**: Identify the rooms.

Functions and other members implemented:

- Constructors, Setters & Getters

2.10 Dungeon Class

The class for the whole Dungeon.

Objects implemented:

- **player**
- **Room** rooms**: My dungeon is a rectangular grid, so it is sufficient to store it in a 2D array.

Functions and other members implemented:

- Constructors
- **void createPlayer(), void linkingRooms(), void createMap(), void initGame()**: Functions for initializing the dungeon. If the user wants to load previous data, the **initGame()** function will call the record system, otherwise it calls the **createPlayer(), createMap()** function. An algorithm is used to generate the map (see 2.12.)
- **void handleMovement()**: Allowing player to move from one room to the other.
- **void chooseAction()**: Allowing player to choose next action.
- **void checkGameLogic()**: Check if the game should end. The player wins when he/she beats the monster and loses when the player has non-positive health.
- **void runDungeon()**: Deal with the whole game process.

2.11 Record Class

Implements the record system.

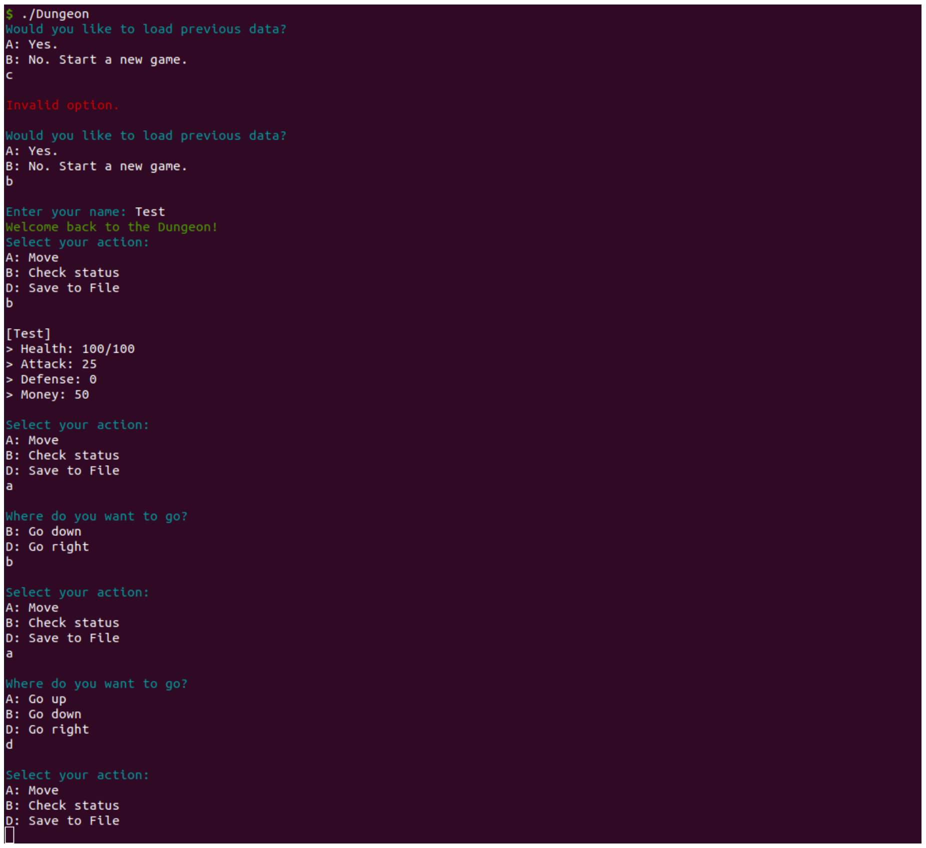
Functions and other members implemented:

- **void load()**: Function for loading data.
- **void record()**: Functions for recording data. The data are stored in the **data** file.

2.12 Other details

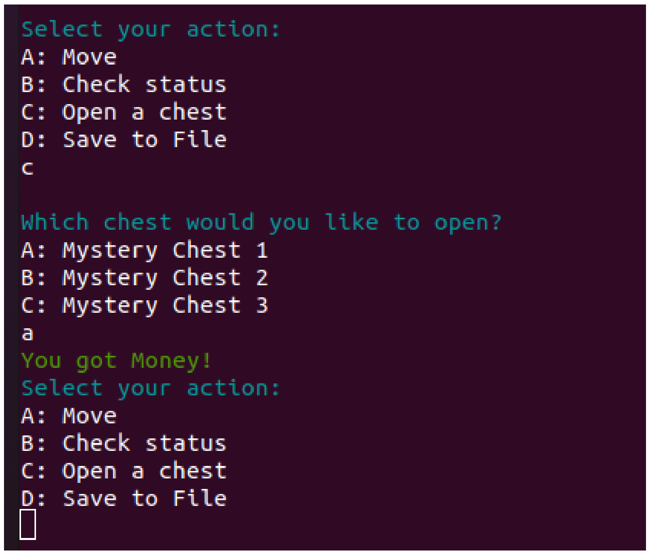
- The algorithm for generating the dungeon is the following:
  1. The default size of the dungeon is 3×4. First put the player in coordinate $(1, 1)$ and monster in coordinate $(3, 4)$.
  2. Then place the chest, the NPC and the monster (Slime) one by one. The places are chosen uniformly at random as long as there is no people in the chosen coordinate.
- For beautifying TUI, I have changed some color of the lines by using ANSI escape codes.

3 Results



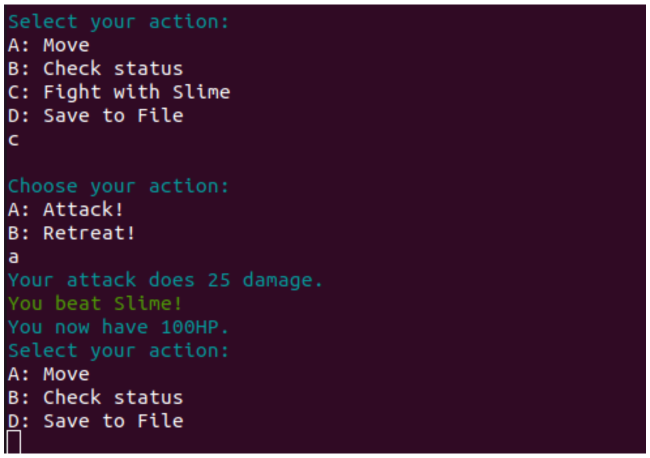
```
$ ./Dungeon
Would you like to load previous data?
A: Yes.
B: No. Start a new game.
c

Invalid option.

Would you like to load previous data?
A: Yes.
B: No. Start a new game.
b

Enter your name: Test
Welcome back to the Dungeon!
Select your action:
A: Move
B: Check status
D: Save to File
b

[Test]
> Health: 100/100
> Attack: 25
> Defense: 0
> Money: 50

Select your action:
A: Move
B: Check status
D: Save to File
a

Where do you want to go?
B: Go down
D: Go right
b

Select your action:
A: Move
B: Check status
D: Save to File
a

Where do you want to go?
A: Go up
B: Go down
D: Go right
d

Select your action:
A: Move
B: Check status
D: Save to File
```

Figure 2: The basic TUI. Error messages are in red; information is in cyan; other messages are in green.



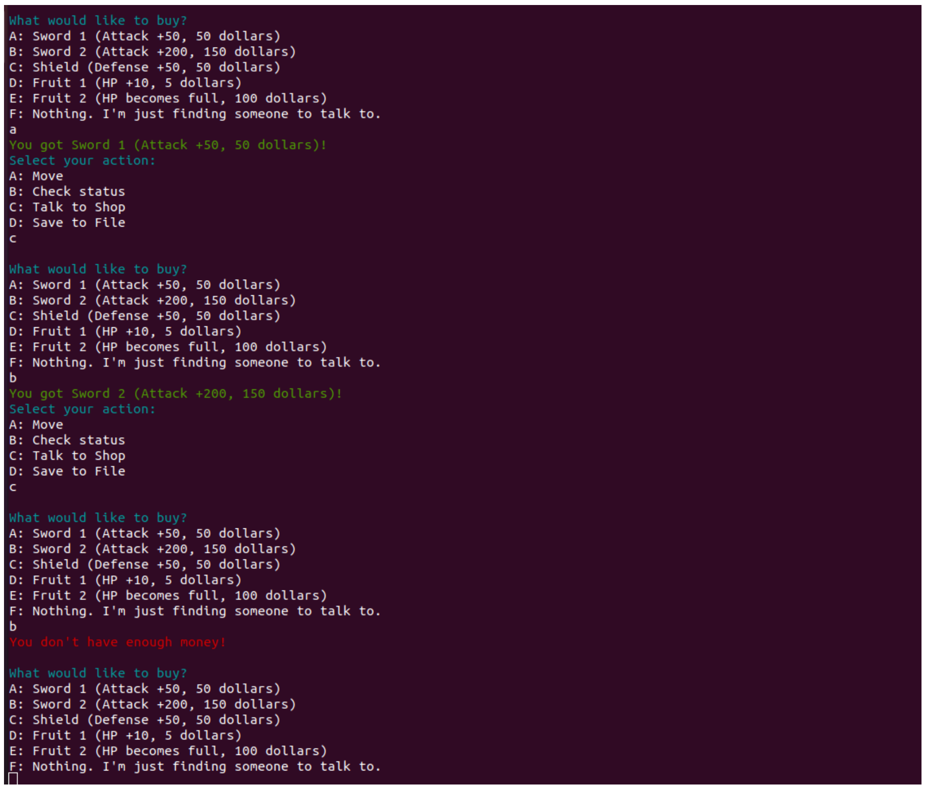
```
Select your action:
A: Move
B: Check status
C: Open a chest
D: Save to File
c

Which chest would you like to open?
A: Mystery Chest 1
B: Mystery Chest 2
C: Mystery Chest 3
a
You got Money!
Select your action:
A: Move
B: Check status
C: Open a chest
D: Save to File
```

Figure 3: Opening chests.



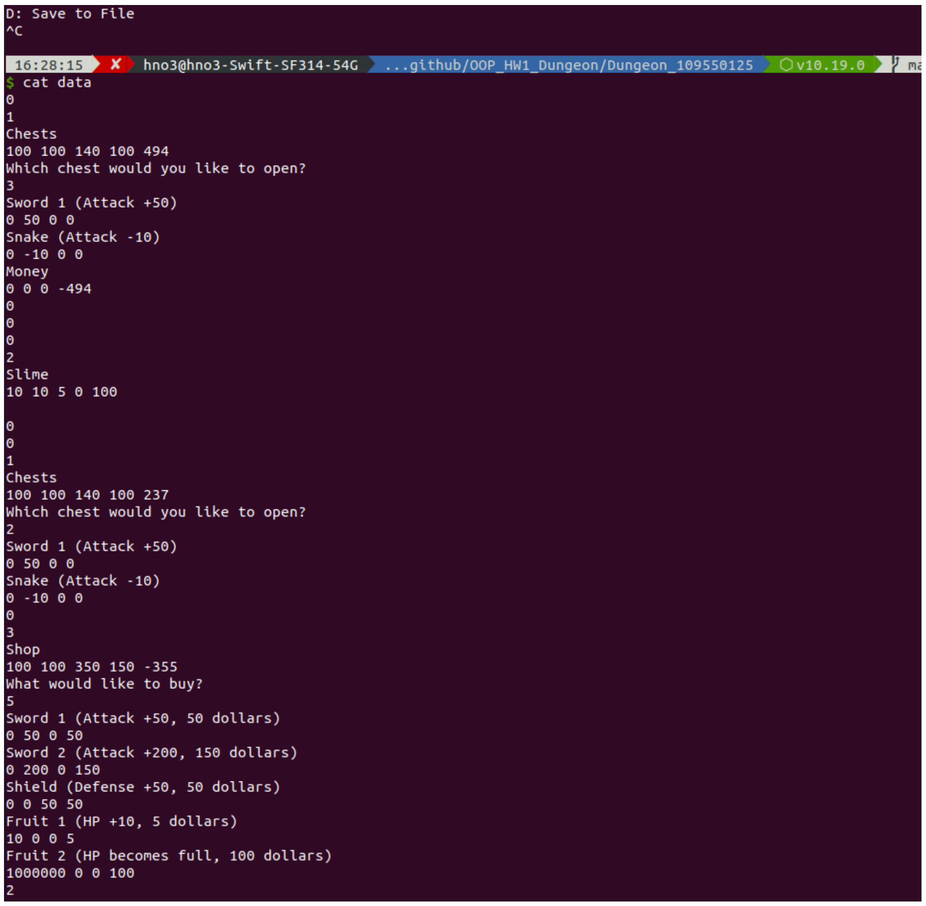
```
Select your action:
A: Move
B: Check status
C: Fight with Slime
D: Save to File
c

Choose your action:
A: Attack!
B: Retreat!
a
Your attack does 25 damage.
You beat Slime!
You now have 100HP.
Select your action:
A: Move
B: Check status
D: Save to File
```

Figure 4: Fighting system.



```
What would like to buy?
A: Sword 1 (Attack +50, 50 dollars)
B: Sword 2 (Attack +200, 150 dollars)
C: Shield (Defense +50, 50 dollars)
D: Fruit 1 (HP +10, 5 dollars)
E: Fruit 2 (HP becomes full, 100 dollars)
F: Nothing. I'm just finding someone to talk to.
a
You got Sword 1 (Attack +50, 50 dollars)!
Select your action:
A: Move
B: Check status
C: Talk to Shop
D: Save to File
c

What would like to buy?
A: Sword 1 (Attack +50, 50 dollars)
B: Sword 2 (Attack +200, 150 dollars)
C: Shield (Defense +50, 50 dollars)
D: Fruit 1 (HP +10, 5 dollars)
E: Fruit 2 (HP becomes full, 100 dollars)
F: Nothing. I'm just finding someone to talk to.
b
You got Sword 2 (Attack +200, 150 dollars)!
Select your action:
A: Move
B: Check status
C: Talk to Shop
D: Save to File
c

What would like to buy?
A: Sword 1 (Attack +50, 50 dollars)
B: Sword 2 (Attack +200, 150 dollars)
C: Shield (Defense +50, 50 dollars)
D: Fruit 1 (HP +10, 5 dollars)
E: Fruit 2 (HP becomes full, 100 dollars)
F: Nothing. I'm just finding someone to talk to.
b
You don't have enough money!

What would like to buy?
A: Sword 1 (Attack +50, 50 dollars)
B: Sword 2 (Attack +200, 150 dollars)
C: Shield (Defense +50, 50 dollars)
D: Fruit 1 (HP +10, 5 dollars)
E: Fruit 2 (HP becomes full, 100 dollars)
F: Nothing. I'm just finding someone to talk to.
```

Figure 5: Trading with NPC.



```
D: Save to File
^C

 16:28:15  ✘  hno3@hno3-Swift-SF314-54G  ...github/OOP_HW1_Dungeon/Dungeon_109550125  ⬡ v10.19.0  ma
$ cat data
0
1
Chests
100 100 140 100 494
Which chest would you like to open?
3
Sword 1 (Attack +50)
0 50 0 0
Snake (Attack -10)
0 -10 0 0
Money
0 0 0 -494
0
0
0
2
Slime
10 10 5 0 100

0
0
1
Chests
100 100 140 100 237
Which chest would you like to open?
2
Sword 1 (Attack +50)
0 50 0 0
Snake (Attack -10)
0 -10 0 0
0
3
Shop
100 100 350 150 -355
What would like to buy?
5
Sword 1 (Attack +50, 50 dollars)
0 50 0 50
Sword 2 (Attack +200, 150 dollars)
0 200 0 150
Shield (Defense +50, 50 dollars)
0 0 50 50
Fruit 1 (HP +10, 5 dollars)
10 0 0 5
Fruit 2 (HP becomes full, 100 dollars)
1000000 0 0 100
2
```

Figure 6: The actual save data when the record system is used.

4 Discussions

Some details and enhancements I think I can do better:

- In the GameCharacter class, I have actually implement the **isDead()** function, but it doesn't work when I am testing (so the checking of game logic is actually $HP \leq 0$.) Learning how to use gdb may help?
- Separate the monster data and the function in the Dungeon class implementation.
- Learn some coding styles and design patterns to write cleaner code.
- Learn how to visualize the game.

5 Conclusion

It is my first time to write a tiny project, and I have learned a lot some it. Hope I can write some larger projects in the future.