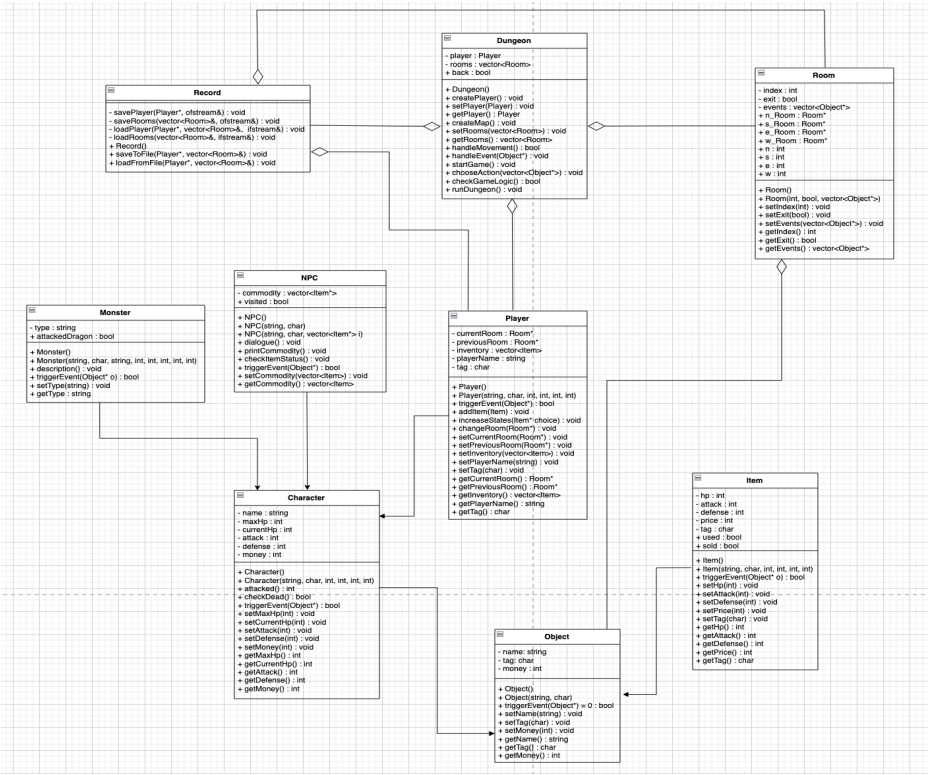
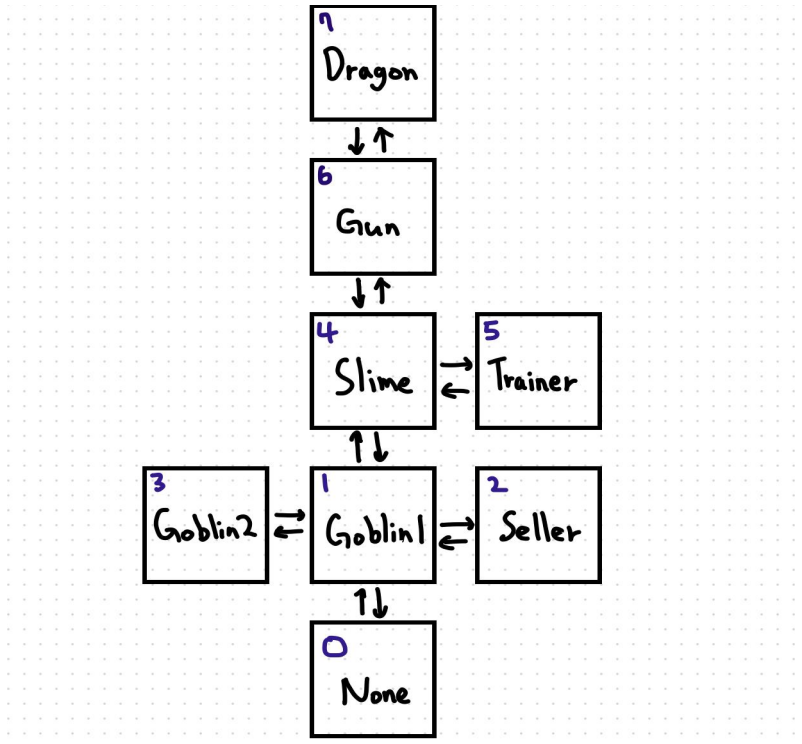
110700049 楊子萱

The Dungeon Report

UML Design



Room Design



110700049 楊子萱

Implementation detailed

1.Room

```
void Dungeon::createMap(){
    vector<Room> rooms(8);
    int link[8][4]={{1, -1, -1, -1}, {4, 0, 3, 2}, {-1, -1, 1, -1}, {-1, -1, -1, 1}, {6, 1, -1, 5}, {-1, -1, 4, -1}, {7, 4, -1, -1}, {-1, 6, -1, -1}};

    //create items
    Item* att=new Item("Attack Training Ticket",'a', 0, 100, 0, 50);
    Item* dtt=new Item("Defense Training Ticket",'d', 0, 0, 100, 50);
    Item* energy=new Item("Energy Drink",'e', 100, 0, 0, 50);
    Item* gun=new Item("Water Gun", 'w', 0, 0, 0,100);
    vector<Item*> commodities={att, dtt, energy};
    //create monsters
    Monster* goblin1=new Monster("Goblin", 'm', "none", 150, 150, 75, 75, 50);
    Monster* goblin2=new Monster("Goblin", 'm', "none", 200, 200, 100, 80, 50);
    Monster* slime=new Monster("Slime", 'm', "water", 400, 400, 80, 200, 50);
    Monster* dragon=new Monster("Dragon", 'd', "fire", 500, 500, 250, 275, 0);
    //create NPC
    NPC* seller=new NPC("Seller", 's', commodities);
    NPC* trainer=new NPC("Trainer", 't');

    vector<Object*>none, gob1(1,goblin1), gob2(1,goblin2), sli(1,slime), dra(1,dragon), sel(1,seller), tr(1,trainer), wg(1,gun);
    //create rooms & set index
    for(int i=0; i<8; i++){
        Room* r= new Room;
        r->setIndex(i);
        r->n=link[i][0];
        r->s=link[i][1];
        r->w=link[i][2];
        r->e=link[i][3];
        rooms[i]=*r;
    }
    //set event to room
    rooms[0].setEvents(none);
    rooms[1].setEvents(gob1);
    rooms[2].setEvents(sel);
    rooms[3].setEvents(gob2);
    rooms[4].setEvents(sli);
    rooms[5].setEvents(tr);
    rooms[6].setEvents(wg);
    rooms[7].setEvents(dra);
    //set exit
    for(int i=0; i<8; i++){
        rooms[i].setExit(0);
    }
    //set links
    for(int i=0; i<8; i++){
        if(rooms[i].n >= 0){
            rooms[i].n_Room=&rooms[rooms[i].n];
        }else {
            rooms[i].n_Room=nullptr;
        }

        if(rooms[i].s >= 0){
            rooms[i].s_Room=&rooms[rooms[i].s];
        }else {
            rooms[i].s_Room=nullptr;
        }

        if(rooms[i].w >= 0){
            rooms[i].w_Room=&rooms[rooms[i].w];
        }else {
            rooms[i].w_Room=nullptr;
        }

        if(rooms[i].e >= 0){
            rooms[i].e_Room=&rooms[rooms[i].e];
        }else {
            rooms[i].e_Room=nullptr;
        }
    }
    setRooms(rooms);
}
```

I used a linked list to create and to link the rooms. First, I declared a vector of type Room to store my rooms. To create links, I declared a 2-D array to store the neighbor room indexes in its north, south, west, east direction. If there is no room in one of the directions, the index -1 is used. Using a for loop, each room is linked to its neighbors by assigning the neighbor room to class Room's member variables ("n_Room", "s_Room", "w_Room", "e_Room"). If the index is -1, a nullptr is assigned.
The events are set to the rooms by first declaring Item, Monster, and NPC objects. Each object is put in separate vectors, except from the Item objects, which are commodities the seller sells, are put together in a vector of type Item*. The events are then passed to the member function "setEvents" of class Room, assigning the events to its respective rooms.

2.Movement

```
bool Dungeon::handleMovement(){
    int option;
    while(1){ //若玩家走到死路就要一直問要去哪
        cout<<"_________________________________\n";
        cout<<"|Where would you like to go ?|\n";
        cout<<"---------------------------------\n";
        cout<<">>1. Go North\n";
        cout<<">>2. Go South\n";
        cout<<">>3. Go West\n";
        cout<<">>4. Go East\n";
        cout<<">>5. I changed my mind\n";
    retry:
        cout<<"I will... ";
        cin>>option;
        if(option==1){
            if(player.getCurrentRoom()->n_Room==nullptr){
                cout<<endl<<"Apparently you've reached a dead end. Try a different direction.\n";
                continue;
            }else{
                player.changeRoom(&rooms[player.getCurrentRoom()->n]);
                cout<<endl<<"Knock. Knock. (Door Opens)"<<endl<<"You're now in room"<<player.getCurrentRoom()->getIndex()<<endl;
                break;
            }
        }else if(option==2){
            if(player.getCurrentRoom()->s_Room==nullptr){
                cout<<endl<<"Apparently you've reached a dead end. Try a different direction.\n";
                continue;
            }else{
                player.changeRoom(&rooms[player.getCurrentRoom()->s]);
                cout<<endl<<"Knock. Knock. (Door Opens)"<<endl<<"You're now in room"<<player.getCurrentRoom()->getIndex()<<endl;
                break;
            }
        }else if(option==3){
            if(player.getCurrentRoom()->w_Room==nullptr){
                cout<<endl<<"Apparently you've reached a dead end. Try a different direction.\n";
                continue;
            }else{
                player.changeRoom(&rooms[player.getCurrentRoom()->w]);
                cout<<endl<<"Knock. Knock. (Door Opens)"<<endl<<"You're now in room"<<player.getCurrentRoom()->getIndex()<<endl;
                break;
            }
        }else if(option==4){
            if(player.getCurrentRoom()->e_Room==nullptr){
                cout<<endl<<"Apparently you've reached a dead end. Try a different direction.\n";
                continue;
            }else{
                player.changeRoom(&rooms[player.getCurrentRoom()->e]);
                cout<<endl<<"Knock. Knock. (Door Opens)"<<endl<<"You're now in room"<<player.getCurrentRoom()->getIndex()<<endl;
                break;
            }
        }else if(option==5){
            back=1;
            chooseAction(player.getCurrentRoom()->getEvents());
        }else{
            cout<<"Please input the above options only.\n";
            goto retry;
        }
        break;
    }
    return 1;
}
```

This “handleMovement” function’s purpose is to ask which direction the player wants to go. By using a while loop, it will keep asking the question until a correct direction is chosen and thus moved to another room. If the player’s input is not in the option list, the goto statement will prompt the user to input again using the valid options only.

The changing of rooms is managed by a member function of class Player named “changeRoom”, which passes in the current room’s neighbor room’s index and sets it to the current room and the old current room is set to the previous room.

3. Actions Menu

```
252 void Dungeon::chooseAction(vector<Object*> o){
253     if(o.empty() || back==1){
254         int option;
255         cout<< "_____________________________\n";
256         cout<<"|Please choose your action.|\n";
257         cout<<"-----------------------------\n";
258         cout<<">>1. Move\n";
259         cout<<">>2. Check my status\n";
260         cout<<">>3. Save and leave game\n";
261     retry:
262         cout<<"I want to... ";
263         cin>> option;
264         if(option==1){
265             handleMovement();
266             back=0;
267         }else if(option==2){
268             player.triggerEvent(&player);
269         }else if(option==3){
270             cout<<"Game record saved.\n";
271             cout<<"See you next time,"<<player.getName();
272             Record record;
273             record.saveToFile(&player, rooms);
274             exit(1);
275         }else{
276             cout<<"Please input the above options only.\n";
277             goto retry;
278         }
279     }else{
280         handleEvent(o[0]);
281     }
282 }
```

The member function, “chooseAction”, of class Dungeon is in charge of asking whether the player wants to move to a different room, check current status, or save current progress and leave the game. This function is called when there’s no event, o.empty() is true, in a room (i.e Room0), or when the player doesn’t want to move and chooses to return to the actions menu from the movement menu, the bool variable back is true.

If the player wants to move, the “handleMovement” function is called and sets the bool variable back to false. If the player wants to check status, the virtual function of class Player is called, which prints out the attributes and characteristics of the player. If the player wants to save the game, an object of class Record is created and the member function “saveToFile” is called, which passes in the address of the player and rooms.

When the player enters a room that has an event, the function “handleEvent” is called with the first element in the vector of class Object* as the argument.

3. Handle Event

```
188 void Dungeon::handleEvent(Object* o){
189     Item* item=dynamic_cast<Item*> (o);
190     Monster* monster=dynamic_cast<Monster*> (o);
191     NPC* npc=dynamic_cast<NPC*> (o);
192     if(item){
193         if(item->used==1){
194             cout<<"You have already picked up the item in the Room.\n";
195         }else{
196             item->triggerEvent(&player);
197         }
198         handleMovement();
199     }else if(monster){
200         if(monster->getCurrentHp()<=0){
201             if(monster->getTag()=='d'){
202                 rooms[7].setExit(1);
203             }else{
204                 cout<<endl<<"You are now in Room"<<player.getCurrentRoom()->getIndex()<<" and you have
                        killed the monster.\n";
205                 handleMovement();
206             }
207         }else{
208             monster->triggerEvent(&player);
209         }
210     }else if(npc){
211         npc->triggerEvent(&player);
212         handleMovement();
213     }else{
214         handleMovement();
215     }
216 }
```

The purpose of this “handleEvent” member function of class Dungeon is to check whether the room entered has an event of a monster, NPC, or Item. This is done by taking the Object pointer as a parameter and by using dynamic cast, which helps determine the type of the event at runtime.

If the event is an Item, the compiler will then check whether the player has picked up the item already or not. If the player has picked up the item, a message of “You have already picked up the item in the Room” will be displayed and the “handleMovement” will be called to ask the user where he/she wants to go. On the other hand, if the player hasn’t picked up the item, the virtual function “triggerEvent” overridden in class Item will be called, which asks the player to pick up the item.

In the event of a monster, when the monster of the current room is still alive, the virtual function “triggerEvent” overridden in class Monster will be called, which asks the player what he/she wants to do with the monster (observe, check player status, attack, or retreat). If the player enters a room where the monster has been defeated, a message telling the player that he/she has killed the monster will be displayed and the “handleMovement” function will be called. On top of that, when the player kills the “Dragon” in the last room (Room7), the bool variable exit will be set to true and the game will be over.

In the event of a NPC, the virtual function “triggerEvent” overridden in class NPC will be called, which asks the player what he/she wants to do with the “Seller” or “Trainer”, depending on which room the player enters. After interacting with the NPC, the “handleMovement” function will be called.

4. Showing Status

```
13  bool Player::triggerEvent(Object * o){
14      Player* player= dynamic_cast<Player*> (o);
15      cout<<endl<<"Name: "<< getName()<<endl;
16      cout<<"Occupation: ";
17      if(getTag()=='k'){
18          cout<<"Knight\n";
19      }else if(getTag()=='a'){
20          cout<<"Archer\n";
21      }else{
22          cout<<"Warrior\n";
23      }
24      cout<<"Hp: "<< getCurrentHp() << "/"<<getMaxHp()<<endl;
25      cout<<"Attack: "<< getAttack()<<endl;
26      cout<<"Defense: "<< getDefense()<<endl;
27      cout<<"Bag: ";
28      if(getInventory().size()){
29          for(int i=0; i<getInventory().size(); i++){
30              cout<<getInventory()[i]->getName()<<" ";
31          }
32          cout<<endl;
33      }else{
34          cout<<"Nothing\n";
35      }
36      return 1;
37  }
```

The purpose of the virtual function “triggerEvent” overridden in class Player is to display the player’s status. This function can be called in the actions menu and in the attack menu when encountering a monster.

First, I used dynamic casting to cast the Object pointer to a Player pointer. If the cast succeeds, the function will then print out the player’s status. The function “getTag” is used to determine the occupation of the player. To print out the inventory, which are the commodities bought by the player from the Seller, I used a for loop. If the player’s inventory is empty, “Nothing” will be printed.

5. Pick Up Items

```cpp
bool Item::triggerEvent(Object *o){
    int option;
    cout<<"_____________________________________________________________\n";
    cout<<"|You see something lying on the foor. Want to pick it up?|\n";
    cout<<"-------------------------------------------------------------\n";
    cout<<">>1. Pick it up\n";
    cout<<">>2. No, I'm scared\n";
    do{
        cout<<"I think I will...";
        cin>>option;
        if (option==1) {
            cout<<endl<<"You found a water gun !"<<endl;
        }else if(option==2){
            cout<<endl<<"Are you sure? You will regret."<<endl;
        }
    }while(option!=1);
    cout<<"Water gun equipped"<<endl;
    used=1;
    return 1;
}
```

```cpp
    //打門結果成功 + 掉金幣
    if(getCurrentHp()<0){
        cout<<endl<<"Oh yay ! You successfully defeated the "<<getName()<<endl;
        if(getName()!="Dragon"){
            cout<<"_____________________________________________________________\n";
            cout<<"|The "<<getName()<< " has dropped some coins. You want to pick it up?"<<setw(6)<<"|\n";
            cout<<"-------------------------------------------------------------\n";
            cout<<">>1. Of course !\n";
            cout<<">>2. I don't feel like picking it up.\n";
            int option;
        retry2:
            cout<<"I think I will...";
            cin>>option;
            if(option==1){
                cout<<endl<<"You picked up 50 coins.\n";
                player->setMoney(player->getMoney() + getMoney());
                cout<<"Your attack and defense has increased by 25.\n"<<"Your HP has recovered and Max HP has increased by 50.\n";
                player->setMaxHp(player->getMaxHp()+50);
                player->setCurrentHp(player->getMaxHp());
                player->setAttack(player->getAttack()+25);
                player->setDefense(player->getDefense()+25);
            }else if (option==2){
                cout<<endl<<"Bad choice. No regrets!\n";
                cout<<"But good news is that your attack and defense has increased by 25.\n"<<"Your HP has recovered and Max HP has increased by 50.\n";
                player->setMaxHp(player->getMaxHp()+50);
                player->setCurrentHp(player->getMaxHp());
                player->setAttack(player->getAttack()+25);
                player->setDefense(player->getDefense()+25);
            }else{
                goto retry2;
            }
        }
    }
```

There’re 2 occasions where the player can pick up items, one of them is when the player enters Room6, and the other is when the player has defeated a monster.

In the former case, the virtual function “triggerEvent” overridden in class Item is in charge of picking up items. The item is a water gun, which can be used in the battle with the “Dragon. The player is forced to pick up the item, if the player chooses not to pick up, the program will keep on asking the player to pick it up. Once the item is picked up, the item is then equipped and the attack with the Dragon is increased by 25.

In the latter case, once the player has defeated the monster, the player can choose to pick up the 50 coins dropped by the monster. If the player chooses to pick up, his/her money will be increased by 50, vice versa.

6. Fighting System

```
bool Monster::triggerEvent(Object *o){
    Player* player=dynamic_cast<Player*>(o);
    int option;
    cout<<endl<<"You entered a "<< getName()<<"'s room.\n";
    cout<<"_____________________________\n";
    cout<<"|What would you like to do?|\n";
    cout<<"-----------------------------\n";
    cout<<">>1. Observe the monster.\n";
    cout<<">>2. Check my status\n";
    cout<<">>3. Attack !\n";
    cout<<">>4. Retreat to previous room !\n";
retry:
    cout<<"I think I will... ";
    cin>>option;
    if(option==1){
        description();
    }else if(option==2){
        player->triggerEvent(player);
    }else if(option==3){
        int round=1;
        //攻擊模式
        while(getCurrentHp()>0 && player->getCurrentHp()>0){
            int pdamage, mdamage;
            if(getTag()=='d'){
                mdamage= (pow(player->getAttack()+25, 2)) / getDefense();
            }else{
                mdamage= (pow(player->getAttack(), 2)) / getDefense();
            }
            pdamage= (pow(getAttack(),2)) / player->getDefense();


            cout<<endl<<"--------------------Round "<<round<<"--------------------"<<endl;
            //Player attack monster
            if(getTag()=='d'){
                cout<<"Launching water attack! ";
            }
            cout<<"You attacked the "<<getName()<<", causing "<<mdamage<<" damage !\n";
            setCurrentHp(attacked(getCurrentHp(), mdamage));
            cout<<getName()<<"'s current HP: "<<getCurrentHp()<<endl;
            if(getCurrentHp()>0){
                //Monster attack player
                cout<<getName()<<" attacked you back, causing "<<pdamage<<" damage !\n";
    //打門結果成功 + 掉金幣
    if(getCurrentHp()<0){
        cout<<endl<<"Oh yay ! You successfully defeated the "<<getName()<<endl;
        if(getName()!="Dragon"){
            cout<<"_____________________________________________________________\n";
            cout<<"|The "<<getName()<< " has dropped some coins. You want to pick it up?"<<setw(6)<<"|\n";
            cout<<"-------------------------------------------------------------\n";
            cout<<">>1. Of course !\n";
            cout<<">>2. I don't feel like picking it up.\n";
            int option;
        retry2:
            cout<<"I think I will...";
            cin>>option;
            if(option==1){
                cout<<endl<<"You picked up 50 coins.\n";
                player->setMoney(player->getMoney() + getMoney());
                cout<<"Your attack and defense has increased by 25.\n"<<"Your HP has recovered and Max HP has increased by 50.\n";
                player->setMaxHp(player->getMaxHp()+50);
                player->setCurrentHp(player->getMaxHp());
                player->setAttack(player->getAttack()+25);
                player->setDefense(player->getDefense()+25);
            }else if (option==2){
                cout<<endl<<"Bad choice. No regrets!\n";
                cout<<"But good news is that your attack and defense has increased by 25.\n"<<"Your HP has recovered and Max HP has increased by 50.\n";
                player->setMaxHp(player->getMaxHp()+50);
                player->setCurrentHp(player->getMaxHp());
                player->setAttack(player->getAttack()+25);
                player->setDefense(player->getDefense()+25);
            }else{
                goto retry2;
            }
        }
    }
    //打門結果失敗 掰掰
    if(player->getCurrentHp()<0){
        cout<<endl<<"Sorry,"<<player->getPlayerName()<<"you are dead. Better luck next time.\n";
        return 0;
    }
    //打敗終極BOSS
    if(getTag()=='d' && getCurrentHp()<0){
        attackedDragon=1;
    }

}else if (option==4){
    player->changeRoom(player->getPreviousRoom());
    cout<<"You are now back to Room"<<player->getCurrentRoom()->getIndex()<<endl;
}else{
    goto retry;
}
return 1;
}
```

The fighting system is managed by the virtual function “triggerEvent” overridden in the class Monster. This is triggered when the player enters a monster’s room. Using dynamic casting to cast the Object pointer to a Player pointer, If the cast succeeds, the function will then ask the player to choose action. Damage is calculated by dividing the square of attack by the enemy’s defense. Attack is launched by the player, if the dragon’s HP isn’t <0, then the monster will attack back. The fight ends when either the player or the monster dies. If the player dies, the game ends. The monster, except from the “Dragon”, will drop coins when defeated by the player. When the “Dragon” is defeated, the bool variable “attackedDragon” is set to true. If the player thinks he/she cannot defeat the monster, he/she can choose to retreat, which changes the current room to the previous room.

110700049 楊子萱

7. NPC

```
void NPC::dialogue(){
    int option;
    cout<<endl<<"You see someone standing in the middle of the room.\n";
    cout<<"_____________________________\n";
    cout<<"|Hello. Do you know who am I ?|\n";
    cout<<"-----------------------------\n";
    cout<<">>1. Magician\n";
    cout<<">>2. Trainer\n";
    cout<<">>3. My DS&OOP Teacher\n";
retry:
    cout<<"I think you are... ";
    cin>>option;
    do{
        if(option==1){
            cout<<" : Magic ~~~ But nope ! Guess again.\n";
            goto retry;
        }else if(option==2){
            cout<<" : OMG! That's right ! How did you know that?\n";
            break;
        }else if(option==3){
            cout<<" : Do I look like him? Guess again. But I hope he scores you an A or above for you final grade.\n";
            goto retry;
        }else{
            cout<<"Please input the above options only.\n";
            goto retry;
```

```
void NPC::printCommodity(){
    vector<Item*> i=getCommodity();
    Item* item=NULL;
    for(int j=0; j<i.size(); j++){
        item=i[j];
        cout<<">>"<<j+1<<". "<<item->getName()<<" ("<<item->getPrice()<<" coins)\n";
    }
    cout<<">>"<<i.size()+1<<". No thanks.\n";
}
```

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

    //Trainer
    if(getTag()=='t'){
        if(visited==0){
            dialogue();
            visited=1;
        }
        //用券換attack, defense
        int option;
    ask:
        cout<<"_____________________________________________________\n";
        cout<<"|What training sessions would you like to have ?|\n";
        cout<<"-----------------------------------------------------\n";
        cout<<">>1. Attack\n";
        cout<<">>2. Defense\n";
        cout<<">>3. No thanks\n";
    retry:
        cout<<"I want to train on...";
        cin>>option;

        Item* choice;
        bool found=0;
        if(option==1){
            for(int i=0; i < player->getInventory().size(); i++){
                if(player->getInventory()[i]->getTag()=='a'){
                    choice= player->getInventory()[i];
                    found=1;
                    cout<<" : Congrats! Your attack has increased by 50 !\n";
                    cout<<" : Good luck on your journey !\n";
                    choice->used=1;
                    player->increaseStates(choice);
                    player->removeItem(i);
                    break;
                }
            }
            if(found==0){
                cout<<"Sorry, seems like you don't have an Attack Ticket.\n";
                cout<<"You can buy one from the Seller.\n";
            }
```

```
        }else if(option==2){
            for(int i=0; i < player->getInventory().size(); i++){
                if(player->getInventory()[i]->getTag()=='d'){
                    choice= player->getInventory()[i];
                    found=1;
                    cout<<" : Congrats! Your defense has increased by 50 !\n";
                    cout<<" : Good luck on your journey !\n";
                    choice->used=1;
                    player->increaseStates(choice);
                    player->removeItem(i);
                    break;
                }
            }
            if(found==0){
                cout<<"Sorry, seems like you don't have a Defense Ticket.\n";
                cout<<"You can buy one from the Seller.\n";
            }
        }else if (option==3){
            cout<<"Well then see you next time!\n";
        }else{
            cout<<"Please input the above options only.\n";
            goto retry;
        }

        cout<<"_____________________________________________________\n";
        cout<<"|Do you want to attend any other training sessions?|\n";
        cout<<"-----------------------------------------------------\n";
        cout<<">>1. Yes please!\n";
        cout<<">>2. No thanks\n";
        cout<<"I think I will...";
        cin>>option;
        if(option==1){
            goto ask;
        }else if(option==2){
            cout<<"Ok then, see you next time!\n";
        }else{
            cout<<"Please input the above options only.\n";
        }
    }
```

```
    if(getTag()=='s'){
        int option;
        cout<<endl<<"Welcome to a hidden store in the Dungeon.\n";
    buy:
        cout<<"_____________________________\n";
        cout<<"|What would you like to buy ?|\n";
        cout<<"-----------------------------\n";
        cout<<"You have "<<player->getMoney()<< " coins.\n";
        printCommodity();

    retry1:
        cout<<"I want to buy...";
        cin>>option;

        if(option == commodity.size()+1){
            cout<<"Ok then. See you next time!\n";
        }else if(option < commodity.size()+1){
            Item* choice = commodity[option-1];
            if(player->getMoney() < choice->getPrice()){
                cout<<" : Oh no, not enough coins. Fight more monsters or find more treasure chests to earn money.\n";
            }else if(choice->sold==1){
                cout<<" : Oh no, the item is sold.\n";
            }else{
                if(choice->getTag()=='e'){ //e for energy drink
                    cout<<" : Good choice ! Here you go ! Good luck on your journey.\n";
                    //售出, 用錢, increase states
                    choice->sold=1;
                    choice->used=1;
                    player->increaseStates(choice);
                    player->setMoney( player->getMoney() - choice->getPrice() );
                    cout<<"You now have "<< player->getMoney() << " coins\n";
                }else if(choice->getTag()=='a'){ //a for attack ticket
                    cout<<" : Good choice ! Here you go ! Good luck on your journey.\n";
                    player->setMoney( player->getMoney() - choice->getPrice() );
                    cout<<"You now have "<< player->getMoney() << " coins\n";
                    //售出, 購入商品到包包
                    choice->sold=1;
                    player->addItem(choice);
                }else if(choice->getTag()=='d'){ //d for defense ticket
                    cout<<" : Good choice ! Here you go ! Good luck on your journey.\n";
                    player->setMoney( player->getMoney()-choice->getPrice() );
                    cout<<"You now have "<< player->getMoney() << " coins\n";
                    //售出, 購入商品到包包
                    choice->sold=1;
                    player->addItem(choice);
                }
            }
```

```
                cout<<"_____________________________________\n";
                cout<<"|Do you want to buy another item?|\n";
                cout<<"-------------------------------------\n";
                cout<<">>1. Yeah sure!\n"<<">>2. No thanks\n";

            retry2:
                cout<<"I think I will...";
                cin>>option;
                if(option==1){
                    goto buy;
                }else if(option==2){
                    cout<<"Okay then. See you next time!\n";
                    //break;
                }else{
                    cout<<"Please input the above options only.\n";
                    goto retry2;
                }
            }else{
                cout<<"Please input the above options only.\n";
                goto retry1;
            }
        }
    return 1;
}
```

There’re 2 NPC in this game, one is a Seller and the other is a Trainer.

When the player enters the Seller’s room (Room2), the player will be asked to buy commodities. If the player doesn’t want to buy, he/she can choose “No thanks” and then the “handleMovement” function will be called. When the player wants to buy commodities, the system will check if the player has enough money, if not, then the system will display a message of not enough money, asking if the player wants to buy another commodity, if not then the “handleMovement” function will be called. The listing of commodity options is managed by the member function “printCommodity”. If the player buys “Energy drink”, his/her maxHp is increased by 100. If the player buys “Attack/ Defense Training Ticket”, the player can use it to exchange with the “Trainer”. When the player successfully buys a commodity, his/her money will be decreased by the price.

When the player enters the Trainer’s room (Room5), if the player has not encountered the NPC before, the “dialogue” function is invoked, which asks the player to guess the identity of the NPC until the player’s guess is correct. If the player has encountered the “Trainer” before, the “dialogue” function will not be invoked. Afterwards, the “Trainer” will ask the player which training sessions to attend. When the player chooses either attack or defense, the player needs to exchange the session with the according ticket. If the player has the ticket, his/her attack or defense will increase by 100 and the ticket will be erased from the player’s inventory.

8. Game Logic

```
284 bool Dungeon::checkGameLogic(){
285     if(player.getCurrentHp()<=0){
286         cout<<"Wish to see you again.\n";
287         cout<<"--------------------GAME OVER--------------------";
288         return 0;
289     }else if(rooms[7].getExit()==1){
290         cout<<"--------------------GAME OVER--------------------";
291         return 0;
292     }
293     else{
294         return 1;
295     }
296 }
297 
298 void Dungeon::runDungeon(){
299     startGame(); //created map and player
300     //Choose actions unless dead or win
301     player.setCurrentRoom(&rooms[0]);
302     while(checkGameLogic()){
303         chooseAction(player.getCurrentRoom()->getEvents());
304     }
305 }
```

The purpose of the “checkGameLogic” function is to end the game when either the player dies (currentHP<=0) or when the bool variable exit of Room7 is equal to true.

The flow of the game is managed by the while loop in the “runDungeon” function.

110700049 楊子萱

9. Character Class Design

```
Character::Character(string name, char tag, int currentHp, int attack, int defense, int money) :Object(name, tag){
    setName(name);
    setTag(tag);
    setCurrentHp(currentHp);
    setAttack(attack);
    setDefense(defense);
    setMoney(money);
}
```

```
void Dungeon::createPlayer(){
    cout<<"_____\n"<< "|Please enter your user name: |\n" << "-----------------------------\n";
    string playerName;
    cout<< "My name is... ";
    cin>>playerName;
    player.setPlayerName(playerName);
    cout<<"Welcome to the Dungeon, "<<player.getPlayerName()<<endl;

    //Choose character
    cout<<"_____\n" << "|Please choose a character. | \n" <<"-----------------------------\n" ;
    cout<<">>1. Knight\n" << ">>2. Archer\n"<<">>3. Warrior\n";
    int character;
retry:
    cout<<"My choice is...";
    cin >> character;
    if(character==1){
        Player player(playerName, 'k', 100, 100, 100, 0);
        cout<<endl<<"A knight is born !\n";
        cout<<"A silver sword is in your hand.\n";
        setPlayer(player);
    }else if(character ==2){
        Player player(playerName, 'a', 85, 85, 85, 0);
        cout<<endl<<"An archer is born !\n";
        cout<<"A bow is in your hand.\n";
        setPlayer(player);
    }else if(character==3){
        Player player(playerName, 'w', 75, 75, 75, 0);
        cout<<endl<<"A warrior is born !\n";
        cout<<"A wooden sword is in your hand.\n";
        setPlayer(player);
    }else{
        cout<< "Please input the above options only.\n" ;
        goto retry;
    }
```

There are 3 occupations, knight, archer, and warrior. The knight is the strongest character, with HP=100, Attack=100, and Defense=100. The second strongest character is the archer, with HP=85, Attack=85, and Defense=85. Warrior is the weakest character, who has HP=75, Attack=75, and Defense=75. Each character has their own weapons, silver sword for knight, bow for archer, and wooden sword for warrior. The weapons are given to the player when they create characters and they use the weapons to defeat monsters. New weapon, water gun, is equipped when the player defeats the slime in Room4 and enters Room6. Since the type of “Dragon” is fire, the water gun can increase damage to the monster, even though the name “water gun” seems weak.

10. Trading System

```
155             if(player->getMoney() < choice->getPrice()){
156                 cout<<" : Oh no, not enough coins. Fight more monsters or find more treasure chests to earn money.\n";
157             }else if(choice->sold==1){
158                 cout<<" : Oh no, the item is sold.\n";
```

```
80              for(int i=0; i < player->getInventory().size(); i++){
81                  if(player->getInventory()[i]->getTag()=='a'){
82                      choice= player->getInventory()[i];
83                      found=1;
84                      cout<<" : Congrats! Your attack has increased by 50 !\n";
85                      cout<<" : Good luck on your journey !\n";
86                      choice->used=1;
87                      player->increaseStates(choice);
88                      player->removeItem(i);
89                      break;
90                  }
91              }
92              if(found==0){
93                  cout<<"Sorry, seems like you don't have an Attack Ticket.\n";
94                  cout<<"You can buy one from the Seller.\n";
95              }
```

As explained in the NPC section, there’re 2 trading places in this game, one with the Seller and one with the Trainer. The system checks whether the player has enough money or has the corresponding item for exchange. For the case of buying commodities with money, this is done by if else statements. If the player’s money < the item’s price, then the player cannot buy the item. Furthermore, if the player has bought the item before, the player cannot buy it again. This is managed by setting the bool variable “sold” to true. For the case of exchanging tickets for the increase in attack or defense, this is done by using a for loop and “getTag” function to search for the right ticket. If found, then the bool variable “used” is set to true and the player’s corresponding attributes are increased. Moreover, the ticket is removed from the player’s inventory.

11. Optional Enhancement

```
5   void Record::saveToFile(Player* player, vector<Room>& rooms){
6       ofstream fplayer("playerRecord.txt");
7       ofstream frooms("roomsRecord.txt");
8       Record::savePlayer(player, fplayer);
9       Record::saveRooms(rooms, frooms);
10  }
```

```
39  void Record::savePlayer(Player* player, ofstream& fplayer){
40      fplayer<<player->getName()<<endl;
41      fplayer<<player->getTag()<<endl;
42      fplayer<<player->getCurrentHp()<<endl;
43      fplayer<<player->getMaxHp()<<endl;
44      fplayer<<player->getAttack()<<endl;
45      fplayer<<player->getDefense()<<endl;
46      fplayer<<player->getCurrentRoom()->getIndex()<<endl;
47      fplayer<<player->getMoney()<<endl;
48  }
49
50  void Record::saveRooms(vector<Room>& rooms, ofstream& frooms){
51      Monster* goblin1= dynamic_cast<Monster*> (rooms[1].getEvents()[0]);
52      Monster* goblin2= dynamic_cast<Monster*> (rooms[3].getEvents()[0]);
53      Monster* slime= dynamic_cast<Monster*> (rooms[4].getEvents()[0]);
54      Monster* dragon= dynamic_cast<Monster*> (rooms[7].getEvents()[0]);
55      frooms<< goblin1->getCurrentHp()<<endl;
56      frooms<< goblin2->getCurrentHp()<<endl;
57      frooms<< slime->getCurrentHp()<<endl;
58      frooms<< dragon->getCurrentHp()<<endl;
59  }
```

```
void Dungeon::startGame(){
    bool start;
    cout<<"_____\n";
    cout<<"|Hello ! Welcome to The Dungeon. Do you accept the challenge ? |" << endl << "|Enter '1' to accept, '0' to decline  (ง'̀-'́)ง " << setw(19) <<"|\n"
        <<"_____\n" ;
    cin >> start;
    if(start==0){
        cout<< "Too bad. See you next time.\n" ;
        exit(0);
    }else if(start==1){
        int option;
        cout<<"_____\n";
        cout<<"|Do you want to start new game or continue from previous game?|\n";
        cout<<"_____\n";
        cout<<">>1. Start new game\n";
        cout<<">>2. Load previous game\n";
    retry:
        cout<<"I want to...";
        cin>>option;
        if(option==2){
            createMap();
            Record record;
            record.loadFromFile(&player, rooms);
            cout<<endl<<"Welcome Back, "<<player.getName()<<endl;
            cout<<"You are now in Room"<<player.getCurrentRoom()->getIndex()<<endl;
        }else if(option==1){
            createMap();
            createPlayer();
        }else{
            cout<<"Please input the above options only.\n";
            goto retry;
        }
    }
}
```

My optional enhancement is the record system. By using fstream and sstream. The player's name, occupation tag, currentHp, maxHp, attack, defense, current room's index, and money are recorded in a text file named "playerRecord" and the monster's currentHp in each room are recorded in the text file "roomsRecord".
Player can choose to save and leave the game in the actions menu. Game is loaded when the player runs the game and chooses to continue playing the previous game.

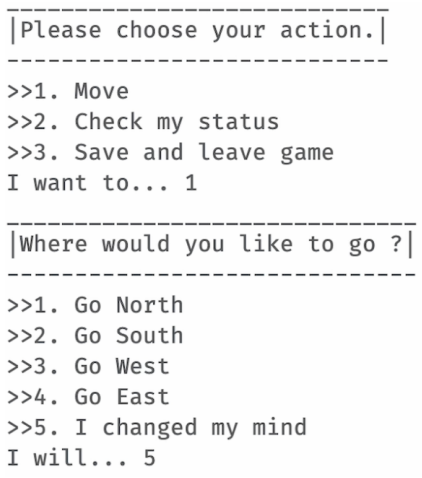
110700049 楊子萱

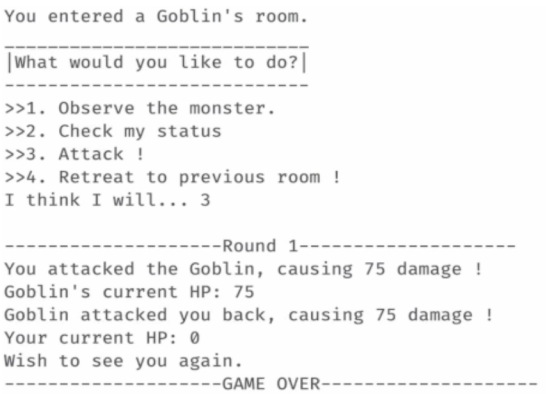
Results

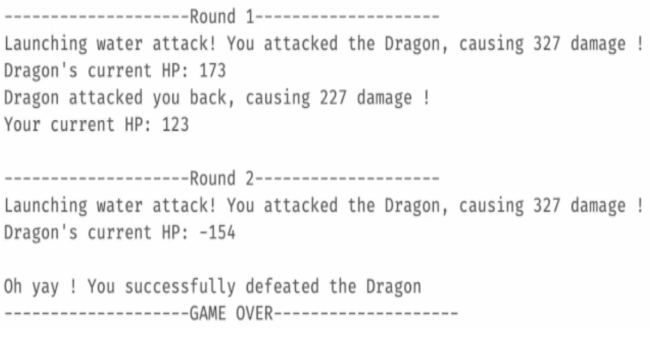
The game's flow starts by asking the player to accept the challenge, start a new or old game, and create a character. When the player explores the different rooms, he/she may encounter a monster, where the player can choose to attack, a NPC to talk to and exchange items, and maybe a treasure(item). Each time the player defeats or finishes interacting with the event in the room, the player will be asked what he/she wants to do or where to go. At last, the game will terminate when the player dies, defeats the dragon, or chooses to save the game.

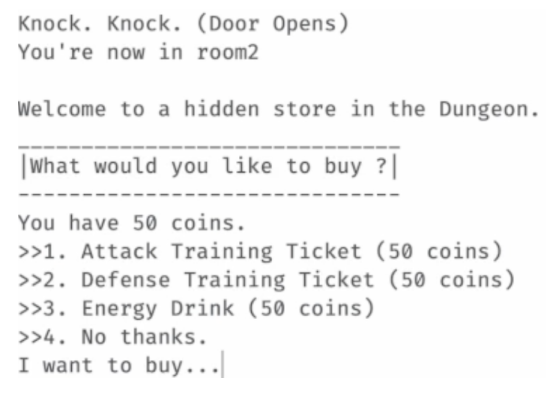
When playing the game, if the player chooses the character archer, he/she may be defeated when encountering the first monster, Goblin, in Room1. However, the other 2 characters should work just fine.

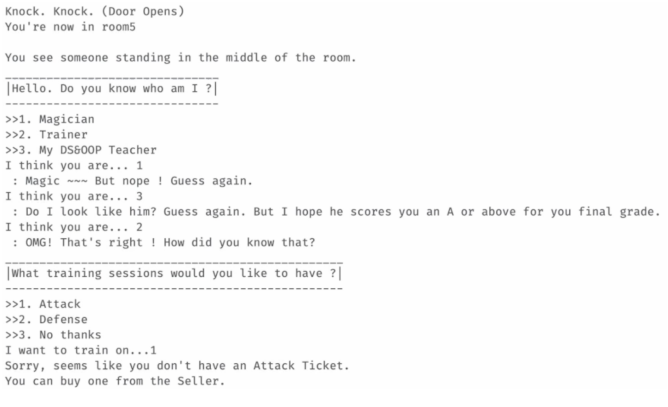
Below are some screenshots of the game, including the start of the game, actions menu, movement menu, game over, victory, buying commodities with seller, and dialogue with trainer.











Discussion

When coding this project, the biggest problem I encountered was setting the current room and changing the rooms. I'd spent hours trying to figure out why the compiler kept on telling me there was a segmentation error. I turned to my friends and they told me to check my declarations of pointers and references again. At last, I found the reason. It was due to my declaration of vector<Room*> instead of vector<Room> in the "createMap" function and I didn't code this->rooms=rooms.

Despite the fact that I found this project quite challenging, I think my programming skills have improved enormously. Through debugging and trying to find out the reasons resulting in the errors, I've gained more knowledge about object-oriented programming, inheritance, and virtual functions. Inheritance allowed me to create derived classes and inherit the properties from the parent class, which allowed me to reuse the code. Polymorphism allowed me to create objects that behave differently based on the context they're used in runtime, and it made the game more dynamic and interactive. Encapsulation allowed the game to be more secure and reliable.

Conclusion

In conclusion, The Dungeon is a game I programmed, whose aim is to defeat the Dragon in the last room. By using OOP, its features such as inheritance, polymorphism, and encapsulation helps to make the game more powerful, secure, and allows the functions to be more dynamic at runtime.

Reflecting on my work, considering the fact that this is my first programming project and game, I think I've tried my very best to make the game run fluently and include the requirements of the project. Some improvements of my game could be to have multiple weapons for each character and the player can choose to use the desired weapon each time he/she encounters a monster. Other improvements could be to make the game more challenging. For example, the player needs to collect all required items in order to unlock the final room.

Despite the fact that there are many more improvements that could be made, I gained a huge sense of achievement since I managed to finish this project on time and I actually quite enjoyed writing my very own first game using OOP.