

The Dungeon Project Report

111550037 嚴偉哲

1 / Implementation

01- Selection List

I designed a function `printScreen` to handle every action that needs player to choose:

```
void Dungeon::printScreen(string title, vector<int> actionList, vector<List> searchPool, void (Dungeon::* func)(int), int mode) {
```

The `searchPool` has all the actions listed, and the `actionList` decides which of them the player can choose. Once selected, it will call the corresponding function `func`.

02- Action Menu

For this, I basically used `actionList` along with `runAction` to achieve this. The former checks the conditions and calls `printScreen`, the latter executes the corresponding function based on the input value.

For example, if the player want to move to another room, the program will call

`actionList(2):`

```
96 case 2: //general
97 {
98     vector<int> listID = { 7, 12 };
99     if (enemyLoad.size() != 0) {
100         listID.push_back(8);
101         listID.push_back(9);
102     }
103     else {
104         if (itemLoad.size() != 0) {
105             listID.push_back(11);
106         }
107         if (npcLoad != NULL) {
108             listID.push_back(10);
109         }
110         listID.push_back(6);
111     }
112     printScreen("Select", listID, list, &Dungeon::runAction, -1);
113     break;
114 }
```

Line 99 – check if there's enemy in the room

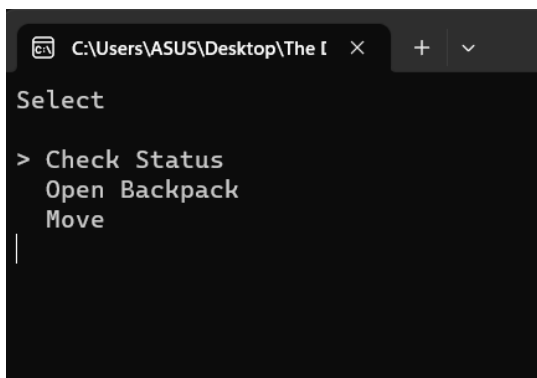
Line 104 – check if there's any treasure

Line 107 – check if there's npc in the room

```
void generateAction() {
    list.push_back({ 0, "New Game" });
    list.push_back({ 1, "Load Game" });
    list.push_back({ 2, "Move North" });
    list.push_back({ 3, "Move South" });
    list.push_back({ 4, "Move East" });
    list.push_back({ 5, "Move West" });
    list.push_back({ 6, "Move" });
    list.push_back({ 7, "Check Status" });
    list.push_back({ 8, "Attack" });
    list.push_back({ 9, "Retreat" });
    list.push_back({ 10, "Talk" });
    list.push_back({ 11, "Open Chest" });
    list.push_back({ 12, "Open Backpack" });
}
```

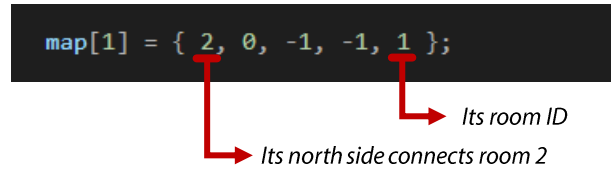
The entire menu action list

This will print out something like this:



03- Movement

For the generation of the whole map, I used array and index to link the rooms together (will be discussed in the discussion and conclusion session). I first numbered all rooms with ids, then type in the direction values (north, south, ...) with the id of other rooms.



To pass the vector list to `printScreen`, it first looks for all four directions to check if there's room connected.

```
case 1: //movement
{
    vector<int> dir;
    for (int i = 1; i <= 4; i++) {
        if (room[player.getRoomID()].getRoom(i) != -1) {
            dir.push_back(i + 1);
        }
    }
    player.setPrevRoom(player.getRoomID());
    hasStayed = false;
    printScreen("Movement", dir, list, &Dungeon::runAction);
    break;
}
```

To move the player to the next room, I simply changed the player's `inRoomID` to the new one. Then it saves the id of the last room to `prev_room`.

```
case 2: //Move North
{
    player.setRoomID(room[player.getRoomID()].getRoom(1));
    break;
}
```

04- Show Status

I created a virtual function `showStats` for the **Entity** class. For the **Player** class, I just print out all the player's information:

The screenshot shows a terminal window with the title "C:\Users\ASUS\Desktop\The I". The output is as follows:

```
Mars Status

Occupation: Knight
Weapon: Basic Sword
Armor: Shield

Health: 100/100
ATK: 40
DEF: 30
LV: 1 (0/10)
Coin: 0G

(press C to exit)|
```

Weapon and **Armor** means what you're equipping now. Will have detailed explanation soon.

05- Pick Up Items

First, I created another virtual function `loadCheck` for the **Entity** class. If the room id matches the `inRoomID` of **Item**, a vector `itemLoad` will add a pointer to the array:

```
void Item::loadCheck(int _id) {  
    extern vector<Item*> itemLoad;  
    if (getRoomID() == _id) {  
        itemLoad.push_back(this);  
    }  
}
```

Then if `itemLoad` is not empty, a new option would appear:

```
C:\Users\ASUS\Desktop\The I  X  +  v  
Select  
  
    Check Status  
    Open Backpack  
> Open Chest  
    Move  
|
```

Select it will call another `actionList`, this time the items show up:

```
C:\Users\ASUS\Desktop\The I  X  +  v  
Collect  
  
> Swift Boost  
    cancel  
  
Bearing this will give you:  
+20 strength  
|
```

If the arrow is on that item, it will use `showStats` to print its information. Collect it will add it to the player's backpack and delete it from the global list.

```
void Dungeon::pickup(int target) {  
    if (target != itemLoad.size()) {  
        player.addItem(*itemLoad[target]);  
        searchDelete(itemLoad[target], itemList);  
    }  
}
```

→ check if the player selected "cancel" and to prevent error

06- Open Backpack & Equip Weapons

When selecting the "Open Backpack" option, the `actionList` will use the player's backpack as the selection list:

```
case 6: //inspect & use item  
{  
    bp.clear();  
    vector<int> bpID;  
    for (int i = 0; i < (*player.getBackpack()).size(); i++) {  
        bpID.push_back(i);  
        bp.push_back({ i, (*player.getBackpack())[i].getName() });  
    }  
    bp.push_back({ (int)bpID.size(), "cancel" });  
    bpID.push_back((int)bpID.size());  
    printScreen("Backpack", bpID, bp, &Dungeon::equip, 3);  
    break;  
}
```

```
C:\Users\ASUS\Desktop\The I  X + v
Backpack
> Basic Sword (equipped)
  Shield (equipped)
  Swift Boost
  Abandoned Sword
  cancel

Equipping this will give you:
+40 strength
|
```

The item description is the same as the *pick up item* selection list. The variable `equippable` decides whether the item can equip to player. Then for the equipping function, I set the `mainWeapon/mainArmor` to point to the item in the backpack:

```
void Dungeon::equip(int target) {
    if (target != (*player.getBackpack()).size()) {
        if ((*player.getBackpack())[target].getEquippable() == 1) {
            player.setWeapon((*player.getBackpack())[target]);
        }
        else if ((*player.getBackpack())[target].getEquippable() == 2) {
            player.setArmor((*player.getBackpack())[target]);
        }
    }
}

void Player::setWeapon(Item& weapon) {
    if (mainWeapon != NULL) {
        updateStatus(-mainWeapon->getHP(), -mainWeapon->getATK(), -mainWeapon->getDEF());
    }
    mainWeapon = &weapon;
    updateStatus(mainWeapon->getHP(), mainWeapon->getATK(), mainWeapon->getDEF());
}
```

07- Fighting System

The enemies again have `loadCheck` function. If `enemyLoad` is not empty, "attack" option will appear, then the enemy list will show:

```
C:\Users\ASUS\Desktop\The I  X + v
Choose target
> The Fallen Warrior
  The Lost Warrior
  cancel

The guardian of the final room.

HP: 150/150
ATK: 80
DEF: 100
|
```

The description below is done by the `showStats` of **Enemy** class. When attacking the target, `takeDamage` computes the net damage:

```
void Character::takeDamage(int dmg) {
    int netDMG = dmg - getDEF();
    if (netDMG < 0) {
        netDMG = 0;
    }
    else if (netDMG > getCurHP()) {
        netDMG = getCurHP();
    }
    setCurHP(getCurHP() - netDMG);
}
```

If the enemy's HP reaches 0, **lootDrop** calculates Exp and coin the player will gain based on the level of the enemy, then it will be removed from the global list. I also used **rand()** to add some randomness to it:

```
void Dungeon::lootDrop(Enemy target) {
    srand(time(NULL));
    int scale = target.getLvl();
    int coinLoot = 10 * scale + rand() % 4 - 1;
    int expLoot = 6 * scale + rand() % 3 - 1;
    printLine("(You gained " + to_string(coinLoot) + " G and " + to_string(expLoot) + " exp)");
    player.updateStatus(0, 0, 0, coinLoot);
    player.gainExp(expLoot);
}
```

If the player kills all enemies in the room, they can move to the next section. But if not, the enemy(s) will instead attack the player:

```
//enemy attack
if (enemyLoad.size() == 0) { → If room clear
    if (player.getRoomID() != 16) {
        printLine("(You have cleared this room)");
    }
    else {
        endGameID = 3;
    }
}
else {
    for (int i = 0; i < enemyLoad.size(); i++) { → i-th enemy's turn
        printLine("enemy " + to_string(i + 1) + "/" + to_string(enemyLoad.size()) + "\n", 1);
        player.takeDamage(enemyLoad[i]->getATK()); → Gets attacked
        int netDMG = enemyLoad[i]->getATK() - player.getDEF();
        if (netDMG < 0) { ... }
        enemyLoad[i]->attackMessage(netDMG, "player");
        if (player.isDead()) { ... }
        else {
            printLine("(You have " + to_string(player.getCurHP()) + " HP left)");
        }
    }
}
```

If there are more than 1 enemy in the room, they will attack the player by turns:

enemy 1/2

(The Fallen Warrior deals you 20 points)
(You have 120 HP left)

enemy 2/2

(The Lost Warrior deals you 20 points)
(You have 100 HP left)

The player also can retreat to the previous room. To do this, I take the player's **prev_room** value, then check all four directions to see if matches:

```
case 9: //Retreat
    if (player.getRoomID() != 16) {
        for (int i = 1; i <= 4; i++) {
            if (room[player.getRoomID()].getRoom(i) == player.getPrevRoom()) {
                runAction(i + 1);
                printLine(script, 15);
                break;
            }
        }
    }
}
```

08- Character Class Design

There are 3 occupations the player can choose:

-Knight HP: 100, ATK: 40, DEF: 30

I want them to be the choice for new players, as this has the relatively balanced status, so you don't need to think too much about the strategy.

Default: Basic Sword/Shield **Ultimate Weapon:** Abandoned Sword

-Archer HP: 100, ATK: 45, DEF: 25

This occupation has the strongest attack of the three, so the player can do some aggressive moves. To nerf this, I lowered their defense a bit, so it might be a little hard to advance if there are more than 2 enemies in one room.

Default: Basic Bow/Leather Armor **Ultimate Weapon:** Poisonous Arrow

-Magus HP: 80, ATK: 40, DEF: 20 (Special: Range Attack)

The magician is able to attack multiple targets at the same time, so I cut their health and defense points. You start off slow, but can show advantage against 2+ enemies.

Default: Basic Wand/Cape **Ultimate Weapon:** Lightning Wand

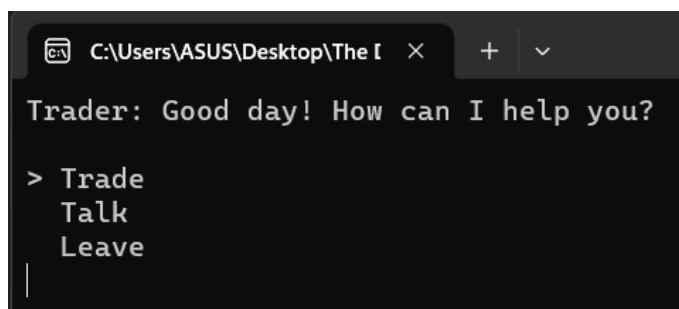
All the corresponding weapons for each occupation can be bought in the shop. If your occupation doesn't match, the ATK points it gives to the player is no more than 20.

However, if it does match, the addATK will be set to 100:

```
void Player::checkMatchedOcc(Item& target) {  
    if (occupation + 1 == target.getSpecialized()) {  
        if (occupation == 0) {  
            target.setATK(100);  
        }  
        else if (occupation == 1) {  
            target.setATK(100);  
        }  
        else if (occupation == 2) {  
            target.setATK(100);  
        }  
    }  
    target.setSpecialized(0);  
}
```

09- NPC & Trading

My design for the npcs contains two actions – **trading** and **chatting**:



The npc is generated with commodities and dialogues. The "Trade" option lists out all the

items the npc has:

```
C:\Users\ASUS\Desktop\The I x + v
Trader: These were entirely grown by myself, they might save you from emergency.

[shelf]
> Canned Fruit
  Abandoned Sword
  Poisonous Arrow
  Lighting Wand
  Reinforced Armor
  cancel

- 15G (You have 44G)

Eating this will give you:
+ 30 health
```

The description below also uses `showStats`, and the npc will have the corresponding script. When buying one item, it will call the `npcSell` function:

```
void npcSell(int item) {
    if (item != (*npcLoad->getShelf()).size()) { check if selected "cancel"
        sell tempItem = (*npcLoad->getShelf())[item];
        vector<Dialogue> script = npcLoad->getDialogue();
        if (dungeon.player.getCoin() < tempItem.price) { if the player has enough money
            printLine(npcLoad->getName() + ": " + script[findDialogue(21)].GetLine());
        }
        else {
            dungeon.player.addItem(tempItem.item);
            dungeon.player.updateStatus(0, 0, 0, -tempItem.price);
            if (!tempItem.permanent) { if the item has infinitely many
                searchDelete(&tempItem, *npcLoad->getShelf());
            }
            if (tempItem.id >= 11 && tempItem.id <= 13) {
                printLine(npcLoad->getName() + ": " + script[findDialogue(20)].GetLine(), 2);
                //check if match
                if (tempItem.id == dungeon.player.getOcc() + 11) { if matches the occupation
                    printLine(script[findDialogue(23)].GetLine());
                }
                else {
                    printLine(script[findDialogue(22)].GetLine());
                }
            }
            else {
                printLine(npcLoad->getName() + ": " + script[findDialogue(20)].GetLine());
            }
        }
    }
}
```

If instead chooses the "Talk" option, it will list all conversation topics the player can pick:

```
C:\Users\ASUS\Desktop\The I x + v
The Gatekeeper: What do you want to know?

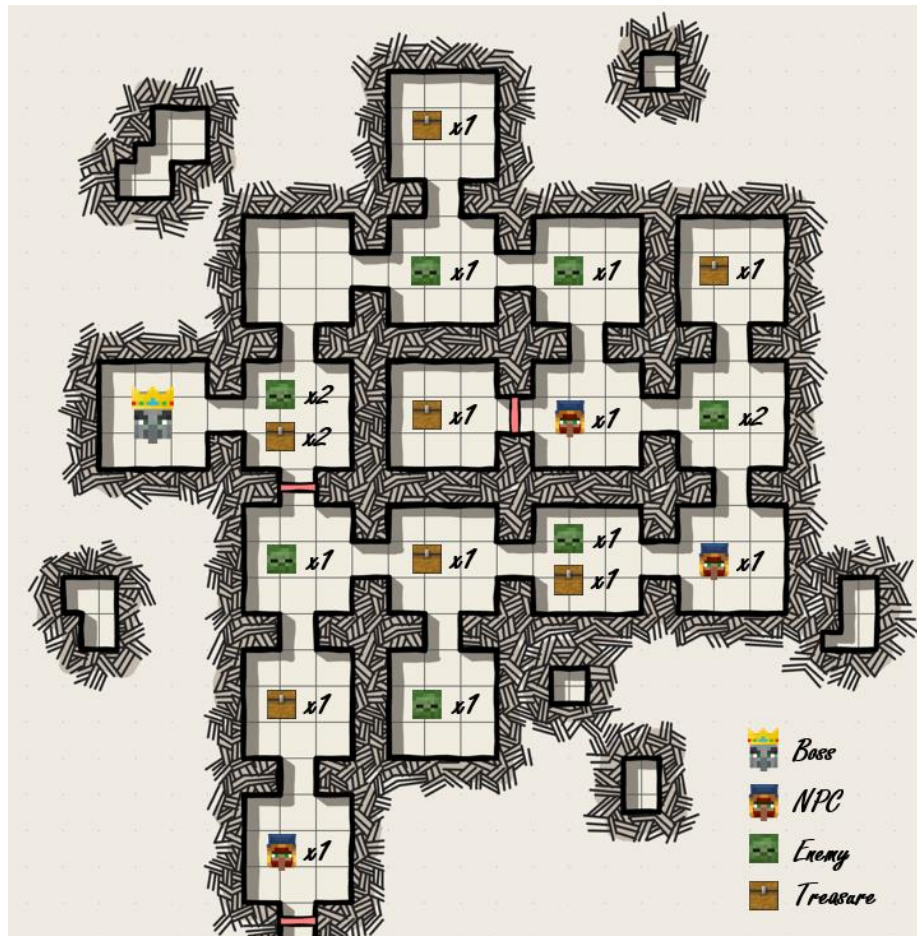
> What's inside here
  Tips for beating this dungeon
  Do you work here
```

Select one of them enters a series of scripts:

```
C:\Users\ASUS\Desktop\The I x + v
The Gatekeeper: I wouldn't call this a job, they don't pay me.
The Gatekeeper: But someone needs to watch the entrance, right?
```

10- Map Design

There's a total of 17 rooms in this dungeon, the design and the entity distribution are shown below:



The red lines are locked doors and need to use different method to access them:

I wrote an additional if-statement to prevent the player from entering the next room and trigger dialogue at the same time:

```
}  
if (title == "Movement" && room[player.getRoomID()].getLockDir() == (actionList[curPos] - 1)) {  
    lockedMsg(player.getRoomID());  
    ...  
}
```



The first one requires the player to go through the dungeon, triggering the boss room dialogue:

```
C:\Users\ASUS\Desktop\The T x + v  
(You heard something unlocked from a distance.)  
|
```




This door has to be unlocked from the north side. It will ask the player to spend 50G to unlock it:

```
C:\Users\ASUS\Desktop\The I x + v
(The door seems to be locked from the other side)
```

from south

```
C:\Users\ASUS\Desktop\The I x + v
(Spend 50G unlocking this door?)
> yes
no
```

from north



The last one requires player to bring the toolbox to Smith (the npc in that room):

```
C:\Users\ASUS\Desktop\The I x + v
Smith: bring me the toolbox back, and i'll give you the key.
```

When talking to Smith, the game checks if the player has the toolbox in their backpack, if true then removes it, and the door unlocks automatically:

```
case 10: //Talk
    if (player.getRoomID() == 9) { → if at the room with Smith
        bool boxGet = false;
        for (int i = 0; i < player.getBackpack()->size(); i++) {
            if ((*player.getBackpack())[i].getName() == "Old Toolbox") { → if the player has the toolbox
                boxGet = true;
                player.removeItem(i);
                break;
            }
        }
        if (boxGet) {
            unlockDoor(0); → trigger cutscene and set the "lockedDir" to 0
        }
    }
    npcAction(0);
    break;
```

11- Health Regeneration

There are 2 ways for player to restore their health point:

- Canned Fruit

This can be bought from the Trader and is a permanent product. Consuming it gains 30 HP.

```
Abandoned Sword (equipped)
> Canned Fruit
Mysterious Potion
cancel

Eating this will give you:
+ 30 health

(press C to eat)
```

description

```
C:\Users\ASUS\Desktop\The I x + v
(Your health has increased to 90)
```

eating message

- Level Increase

When leveling up, player's maximum health will increase, resulting in the growth of current health point.

```
int upgrade = (int)(getLvl() * 0.5 * getLvl() * 0.5);
updateStatus(20 * upgrade, 15 * upgrade, 10 * upgrade);
```

the formula for the level upgrade

12- Player Level/EXP System

After killing an enemy, the player will gain some experience (EXP). When the EXP reaches a certain amount, the player's level (LV) will increase by 1.

The diagram illustrates the level-up logic. On the left, the `Player::gainExp` function is shown. It calculates the next level requirement and sets the level when the current experience is reached. A red arrow points from the `setLvl(1);` line to the `nextLvlReq` function on the right. A blue bracket groups the level-setting logic with the annotation "sets level every time". A red arrow points from the `if (cmp != getLvl())` condition to the annotation "if the level has changed". The `nextLvlReq` function on the right shows the required experience for each level.

```
void Player::gainExp(int _exp) {
    int cmp = getLvl();
    exp += _exp;
    if (exp < nextLvlReq(1)) {
        setLvl(1);
    }
    else if (exp < nextLvlReq(2)) {
        setLvl(2);
    }
    else if (exp < nextLvlReq(3)) {
        setLvl(3);
    }
    else if (exp < nextLvlReq(4)) {
        setLvl(4);
    }
    else {
        exp = nextLvlReq(4);
        setLvl(5);
    }
    if (cmp != getLvl()) {
        int upgrade = (int)(getLvl() * 0.5 * getLvl() * 0.5);
        updateStatus(20 * upgrade, 15 * upgrade, 10 * upgrade);
        if (getLvl() == 5) { ... }
        else { ... }
    }
}

int nextLvlReq(int _lvl) {
    if (_lvl == 1) {
        return 10;
    }
    else if (_lvl == 2) {
        return 30;
    }
    else if (_lvl == 3) {
        return 60;
    }
    else if (_lvl == 4) {
        return 100;
    }
    else {
        return 100;
    }
}
```

13- Magus Range Attack

Magus uses range attack, that is, they can hit all enemies in the room in one go.

The diagram shows the `Dungeon::attack` function. It first attacks the main target, then iterates through all enemies in the room. A red arrow points from the `attackEnemy(target, player.getATK(), dead);` line to the annotation "attack main target first". Another red arrow points from the `0.75` value in the `attackEnemy(i, (int)(0.75 * (float)player.getATK()), dead);` line to the annotation "others receive a weaker attack".

```
void Dungeon::attack(int target) {
    vector<int> dead;
    if (target != enemyLoad.size()) {
        //player attack
        attackEnemy(target, player.getATK(), dead);
        if (player.getOcc() == 2) { //magus
            for (int i = 0; i < enemyLoad.size(); i++) {
                if (i != target) {
                    attackEnemy(i, (int)(0.75 * (float)player.getATK()), dead);
                }
            }
        }
    }
}
```

14- Event Cutscenes

When player enters a certain type of rooms or tries to move to a locked room, they may trigger a cutscene.

I added a `hasStayed` variable that determines if the player "enters" the room or "stays" (this is to prevent playing the cutscene over and over again).

Also, a vector list `roomVisited` to monitor which rooms the player has entered (this is to switch to different cutscenes and implement one-time script).

```

for (int i = 0; i < enemyList.size(); i++) { ... }
for (int i = 0; i < itemList.size(); i++) { ... }
for (int i = 0; i < npcList.size(); i++) { ... } } load entities
visited = false;
for (int i = 0; i < roomVisited.size(); i++) { ... }
if (!visited) {
    roomVisited.push_back(player.getRoomID());
}
if (!hasStayed) { → if the player "enters" the room
    //has enemy?
    if (enemyLoad.size() != 0) {
        if (player.getRoomID() != 16) { → if not the boss room
            if (!visited) {
                printLine(script, rand() % 5 + 8, 1);
                printLine("It seems like a fight is inevitable");
            }
            else {
                printLine(script, rand() % 2 + 13);
            }
        }
    }
}
}

```

one of the cutscene, plays when the player enters a room with enemy(s)

There are other situations that will trigger a dialogue. Like entering the shop or boss room for the first time.

When the player moves towards a direction that is either locked or has trigger, the game will call the **lockedMsg** function:

```

void Dungeon::lockedMsg(int room) {
    if (room == 2) { ... }
    else if (room == 9) {
        if (!r9_DialogueTrigger) { → functions like the "visited" variable
            printLine(script, 17);
            printLine(script, 18);
            printLine(script, 51);
            printLine(script, 52);
            printLine(script, 53);
            r9_DialogueTrigger = true;
        }
        else {
            printLine(script, 19);
        }
    }
}

```

15- Game Logic

The game will keep looping back to the action menu until **endGameID** is not 0:

```

void Dungeon::RunGame() {
    generateAction();
    script = addScript(); } initialization
    actionList(0); → the main menu
    while (endGameID == 0) {
        eventCheck(); → handles cutscenes
        hasStayed = true;
        actionList(2); → the action menu
    }
    endingCutscene();
}

```

The game has 3 endings:

- Dead

The player dies. Get by fighting with enemy(s) and lose.

```
else {
    for (int i = 0; i < enemyLoad.size(); i++) {
        println("enemy " + to_string(i + 1) + "/" + to_string(enemyLoad.size()) + "\n", 1);
        player.takeDamage(enemyLoad[i]->getATK());
        int netDMG = enemyLoad[i]->getATK() - player.getDEF();
        if (netDMG < 0) {
            netDMG = 0;
        }
        enemyLoad[i]->attackMessage(netDMG, "player");
        if (player.isDead()) {
            println("(You lost the fight.)");
            endGameID = 1;
            break;
        }
        else {
            println("(You have " + to_string(player.getCurHP()) + " HP left");
        }
    }
}
```

The cutscene:

```
C:\Users\ASUS\Desktop\The I x + v
(Even though you want to keep moving, your body can't take it anymore.)
(Perhaps there has more improvements you can do, perhaps it's not the time yet.)
(You will do it better.)
Ending 1/3 Dead
```

- Chicken Out

The player flees. Get by going all the way to the boss room, then escape from the entrance of the dungeon.

```
void Dungeon::lockedMsg(int room) {
    if (room == 2) { ... }
    else if (room == 9) { ... }
    else if (room == 15) { ... }
    else if (room == 16) { ... }
    else if (room == 0) { ... }
    else if (room == 100) {
        endGameID = 2;
    }
    else if (room == 115) { ... }
}
```

I used the door trigger to achieve this ending, it's 100 because the locked and the trigger event share the same function. So, I used **100+roomID** to handle this.

The cutscene:

```
C:\Users\ASUS\Desktop\The I x + v
(You noticed the entrance is wide open and not even hesitated.)
(This, despite being a galaxy brain move, is not how you supposed to play.)
(However, to award this kind of bravery(of challenging the game design), I must give you some credit.)
(Have a nice day.)
Ending 2/3 Chicken Out
```

- Conquer

The player wins. Get by defeating the boss.

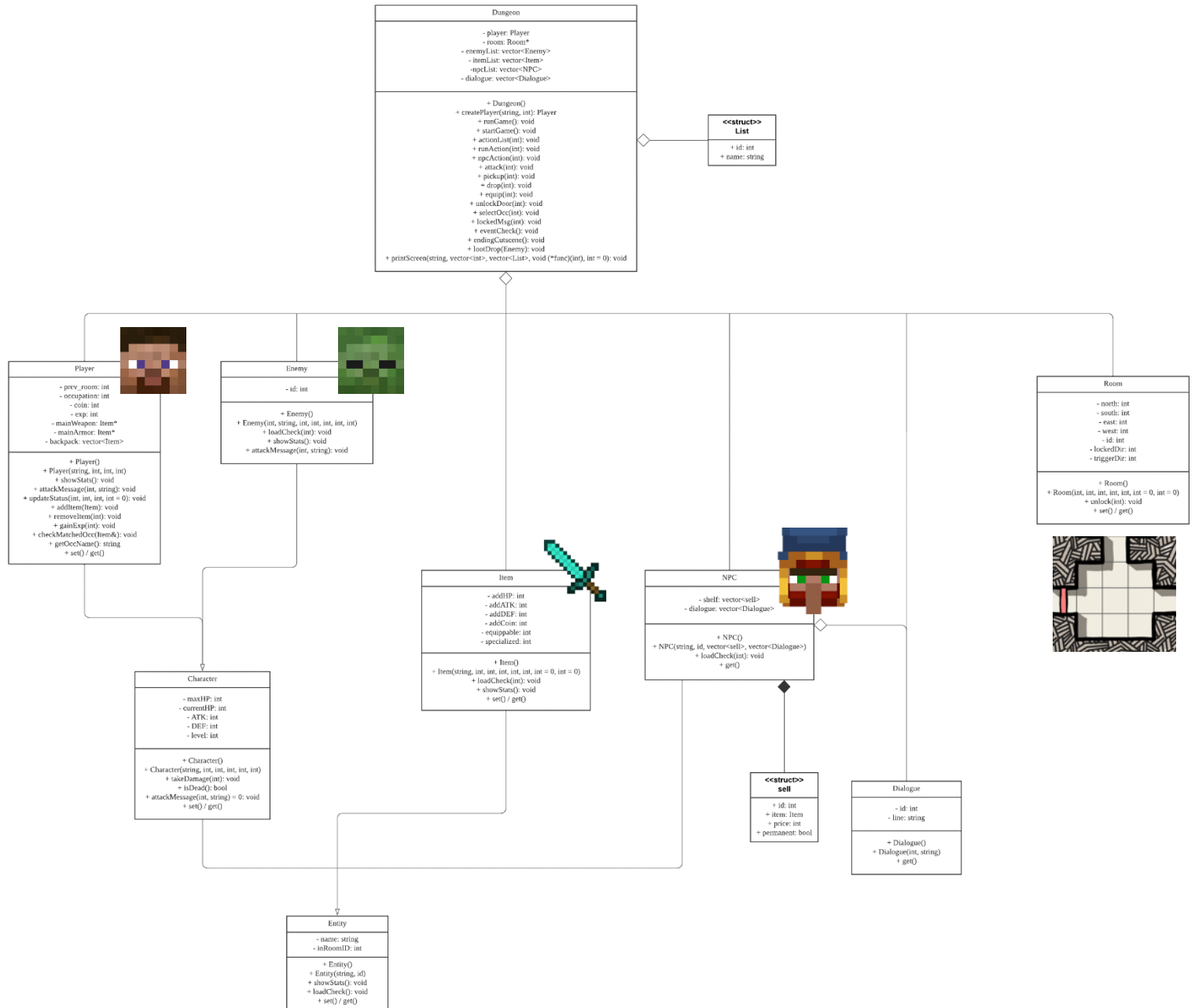
```
//enemy attack
if (enemyLoad.size() == 0) { → if all enemy dies
    if (player.getRoomID() != 16) { → if at boss room
        println("You have cleared this room");
    }
    else {
        endGameID = 3;
    }
}
```

The cutscene:

```
C:\Users\ASUS\Desktop\The I × + v
(Krux fell onto the ground, struggling to stand up.)
(He then teleported out as you tried to give him one last hit.)
(I guess you can count this as a win.)
(You have finally conquered this dungeon.)

(Thanks for playing)
Ending 3/3 Conquer
```

2 / UML



3 / Results

My original idea was to make a story/dialogue-rich rpg, so I wrote a bunch of dialogues into the game.

```
C:\Users\ASUS\Desktop\The I  X  +  v
(You peeked through the entrance, finding a hostile creature)
(It seems like a fight is inevitable)
```

*happens when the player enters a room with enemy(s) **for the first time***

```
C:\Users\ASUS\Desktop\The I  X  +  v
(You decided that you are not ready, and backed out)
```

*happens when the player **retreats***

```
C:\Users\ASUS\Desktop\The I  X  +  v
(You take back your courage and fight once again)
```

*happens when the player **reenters** the room with enemy(s)*

```
C:\Users\ASUS\Desktop\The I  X  +  v
(This room is oddly cozy, even has some food smell.)
(You suddenly realized you haven't eaten for a while.)
```

happens when the player enters the shop

I also attempted to add some personalities to the npcs, like the Trader is a friendly, talkative person, or Smith is cold and somewhat humorous.

```
C:\Users\ASUS\Desktop\The I  X  +  v
Smith: that door? yeah i locked it to prevent, y'know, theft.
Smith: and i locked the key in a toolbox to prevent, y'know, theft.
Smith: and i lost the toolbox.
Smith: brilliant me.
Smith: if you find it, i'll give you what's inside.
```

when the player tries to enter the locked room next to Smith

The second thing I aimed to do is to improve the gameplay and playing experience.

The most apparent one is probably the UI. I cleared the screen everytime the player makes a choice or the dialogue shows, so the screen is cleaner. For the actual selection, I used unbuffered input to make the user choose

and see the information without pressing enter.

In the fighting session (and basically all menu involves entities), I put the enemy's stats below, so that the player can have a better understanding of the situation, and they can make decision and tactics easier. And I wanted the enemy to look not so bland, so I added a line of description:

```
C:\Users\ASUS\Desktop\The I  x + v
Choose target
> Stone Wall
cancel

He won't let you pass that easily

HP: 75/75
ATK: 20
DEF: 120
|
```

the attack menu

```
C:\Users\ASUS\Desktop\The I  x + v
(And select one occupation you prefer. Choose wisely as this is NOT invertible.)

Knight
> Archer
Magus

The ruthless shooter. They're born with deadly precision, but beware not to get flanked.
You will begin with -
HP: 100
ATK: 45
DEF: 25
|
```

the occupation selection menu

```
C:\Users\ASUS\Desktop\The I  x + v
Backpack

Basic Sword (equipped)
Shield (equipped)
Swift Boost
> Abandoned Sword
Canned Fruit
Canned Fruit
cancel

Equipping this will give you:
+100 strength

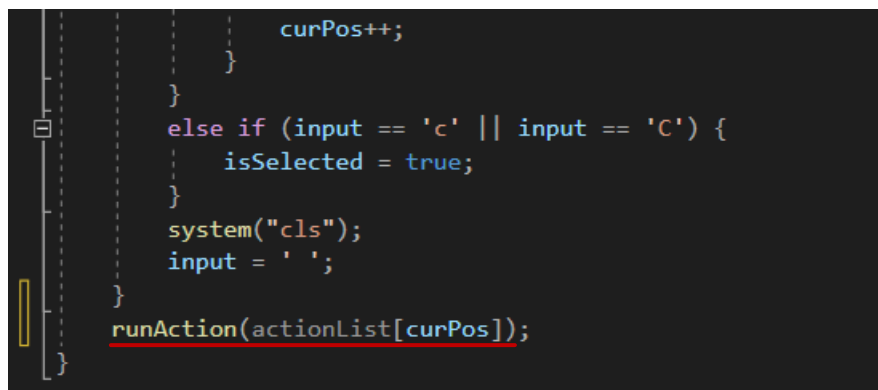
(press C to equip)|
```

the backpack

4 / Discussion

The biggest obstacle I met when coding this game is probably constructing the rooms. I originally used linked list with array, but it keeps compiling error. Then I tried linked list with dynamic array and index with dynamic array, eventually settled with index with array. Even though the problem was solved, I still think the solution is a bit messy. I suspect the issue at the beginning is using the wrong return type, then probably something to do with the properties of dynamic array, and I think this is the direction I should look into to improve this.

Then there's the `printScreen` function. This itself has no big deal, but the code is really long. So, when I realized that I needed to call other functions for selection, my first thought was to copy the function and modify the end part.



```
        curPos++;
    }
}
else if (input == 'c' || input == 'C') {
    isSelected = true;
}
system("cls");
input = ' ';
}
runAction(actionList[curPos]);
```

the original code uses the runAction as the selection call function

That would be no issues if there weren't 5 different call function I need to implement. So, I started to think, if there's a way to change this through function argument. After reading some books, I found that I could use function pointer to achieve that.

Then near the playtesting stage of the game, I noticed that when killing an enemy, the vector list will delete an element and reallocate, but the pointer in enemyLoad does not move with it, so it may point to a wrong enemy and calculate the wrong attack damage. My initial solution is to:

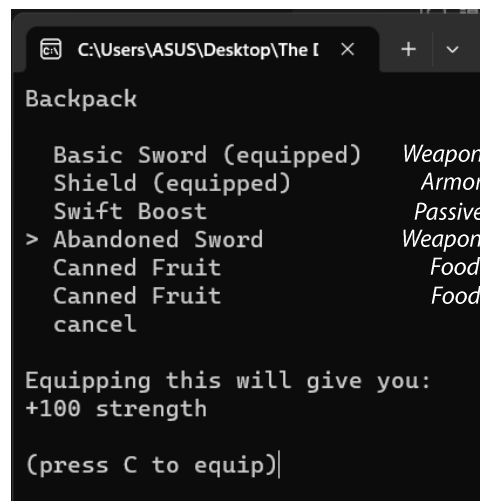
- 1) Save the enemy's name to a temporary variable
- 2) Delete the element in the enemy list
- 3) Search the newly reallocated enemy list to find the enemy
- 4) Pointer points to the new address

But I find it too complicated, so I simply just reload the enemy list, and it seems to work nicely.

Overall, I think this is a very nice chance for me to practice OOP and have some grasp on the game design. At first, I struggled a lot to understand and use the class objects, and I did an immense amount of research just to implement one idea. But as the time goes on, it's just a matter of when until I finished all my goals. This result is not perfect for me, I do think my code can be improved by a lot (like the room construction and dialogue categorization), but given that I only have around 1 month to finish from start, I'm definitely satisfied with this outcome.

5 / Conclusion

To sum up, in this dungeon game, aside from the basic requirements, I designed story elements in it and aimed to improve the player's game experience. If I'll improve and work on this project in the future, I want to first focus on rewriting the code to make it cleaner (and possibly runs faster), then I plan on expanding the dungeon (in fact, the second half of the very original map was discarded) and letting some enemies have special abilities. Lastly, I will improve the UI a bit more, such as the player's backpack, it would be nicer if the items are well-organized.



6 / Video Link

<https://youtu.be/Mh8DPcvFXnQ>