HW3 보고서

로빛 20기 인턴 2025407006 모시온

목차

1. hw3 프로젝트 개요

각각 플레이어, 몬스터 Class 를 나눈 후 실제 예시자료의 게임(Monster Hunter)과 같이 공격과정을 거친 후 도망가는 몬스터를 추적하는 헌터와 이에 대응하여 도망과 반격을 병행하는 몬스터를 제작하여 사냥하는 과정을 작성하는 것이다.

%%과제 공지 사항을 참고하였을 시, 플레이어의 Class 의 멤버변수로서 HP 가 존재하나 동작 예시에서는 이가 구현되는 점을 확인하지 못하여 임의로 가정하여 과제에 공지된 모든 멤버 변수 및 함수를 활용하도록 하였습니다.%%

%%또한 몬스터의 도주를 구현하기 위하여 현재 제시된 몬스터의 초기 좌표 멤버 변수(5,4)를 참고하여 전체 5*5 사이즈의 지도를 임의로 가정하여 플레이어의 몬스터 추적 후 공격 -> 몬스터의 반격 여부 및 도주 -> 추적 -> 엔딩 의 시나리오가 이루어질 수 있도록 하였습니다.%%

%%도주하는 몬스터에 대한 정보 접근성에 관하여입니다. 본래 의도는 몬스터의 좌표를 알려주지 않고 플레이어가 직접 한곳 한곳을 모두 확인하는 방식을 통해 구현하고자 하였으나, 예시 게임을 참고하였을 때 제공되는 지도의 기능을 대신하고자 몬스터의 좌표를 알려주며, 몬스터의 이동을 한 칸으로 제한하여 전체적인 게임의 진행 속도를 증가시킬 수 있도록 하였습니다.%%

2. 알고리즘 설계

2-1. .cpp 설명

```
#include "hw3.hpp"
```

플레이어와 몬스터의 Class 및 멤버 변수와 함수가 저장된 헤더파일을 불러옵니다. 이때, 해당 헤더파일 내부에 iostream 호출이 포함되어 있으므로 추가적인 호출은 생략하였습니다.

```
using namespace HW3;
```

헤더파일에서 설정한 namespace 를 불러오며 해당 main.cpp 이 다른 외부인에 관여에 대해 충돌 저항성을 지니도록 하였습니다. 또한 이름을 HW3 으로 지정하며, 위 프로젝트와 관련된 인물은 모두 해당 코드의 의도를 직관적으로 이해할 수 있도록 하였습니다.

```
srand((unsigned int)time(NULL));


//초기 상태 초기화
Player player(0, 0);
Monster monster(5, 4, 50);

char control;
```

몬스터의 도주 방향을 랜덤으로 지정하기 위하여 사용되었습니다. 겹치는 중복을 피하고자 hw3.hpp 에 포함된 ctime 호출을 통하여 시간에 따른 시드와 그에 관련한 난수 생성을 통해 중복을 예방하였습니다.
이후 과제에 안내된 사항과 동일하게 플레이어의 좌표와 몬스터의 좌표 및 체력을 초기화하였습니다.

이후 사용자의 메뉴 입력을 저장할 control 선언합니다.

```
while (true) {
    if (player.HP <= 0) {
        std::cout << "Player Die !" << std::endl;
        break;
    }
    if (monster.HP <= 0) {
        std::cout << "Moster Die !" << std::endl;
        break;
    }

    std::cout << "Type Command(A/U/D/R/L/S): ";
    char input[6];
    std::cin >> input;
    control = input[0];
    std::cout << std::endl;
```

플레이어 사망, MP 소진, 몬스터 사망 위 세가지의 경우 중 하나가 성립할 때까지 반복되는 while 문에서 각각의 조건문을 통해 플레이어의 체력, 몬스터의 체력을 검토합니다.
이후 총 6 가지의 주어진 명령어의 개수에 맞추어 선언한 문자열 배열 input 을 통해 사용자의 입력을 임시적으로 저장할 수 있도록 하였습니다.
이는 추후 명령어의 복수 입력을 예방하며 추가적으로 올바른 입력이 아닌 경우 예외처리하기 위함입니다.
우선 사용자의 입력을 input 에 모두 저장합니다.
이후 가장 첫번째 인덱스의 요소만 control 에 저장하여 한글자만 인식할 수 있도록 하였으며,

```
//대소문자 관계없이 입력
switch (control) {
    case 'A': case 'a':
        player.Attack(monster);
        break;
    case 'U': case 'u':
        player.Y_move(1);
        std::cout << "Y Position 1 moved!" << std::endl;
        break;
    case 'D': case 'd':
        player.Y_move(-1);
        std::cout << "Y Position -1 moved!" << std::endl;
        break;
    case 'R': case 'r':
        player.X_move(1);
        std::cout << "X Position 1 moved!" << std::endl;
        break;
    case 'L': case 'l':
        player.X_move(-1);
        std::cout << "X Position -1 moved!" << std::endl;
        break;
    case 'S': case 's':
        player.Show_status();
        break;
    default:
        std::cout << "Invaild Input" << std::endl;
}
```

다음의 control 에 따른 switch-case 문을 통해 소문자, 대문자 모두 입력이 가능한 환경을 조성한 후 정해진 명령어가 아닌 모든 입력을 default 로 넘겨 올바르지 못한 입력 값임을 안내할 수 있도록 하였습니다.

control 이 각 주어진 명령어의 문자와 일치하는 경우 hw3.hpp 에 정의된 각 멤버함수의 호출을 통해 게임의 전반적인 진행을 이루게 됩니다.

2-2. .hpp 설명

```
#include <iostream>
#include <cstdlib>
#include <ctime>
```

기본 입력 및 출력을 위한 iostream 과 중복을 일으키지 않는 난수 생성을 위한 cstdlib, ctime 모듈을 불러옵니다.

```
namespace HW3 {
```

namespace 를 HW3 으로 지정하며 이후 과제에 공지된 구성과 동일하게 플레이어와 Monster 의 구성 정의하였습니다.
우선 몬스터의 Class 및 멤버 변수와 함수를 알아보도록 하겠습니다.

```
class Monster {
public:
    int HP = 50;
    int x = 5, y = 4;
    int M_dmg = 33;
    Monster();
    Monster(int x, int y, int HP){
        this -> HP = HP;
        this -> x = x;
        this -> y = y;
        this -> M_dmg = 33;
    }


    int Be_Attacked(int dmg) {
        HP -= dmg;
        if (HP < 0) HP = 0;
        return HP;
    }
```

체력, 좌표, 여기에 추가적인 몬스터의 공격력을 각각의 멤버 변수로 선언하여 초기화하였습니다.
이후 기본 생성자와 함께 내부 멤버 함수에서 값이 지정될 수 있도록 this ->문을 활용하여 초기화를 하였습니다.

바로 아래의 멤버 함수는 몬스터가 공격을 받을 때 사용되며, 플레이어의 공격력을 매개변수로 받아 몬스터 체력에 가감합니다. 이때, 체력 값이 음수인 경우 0 으로 처리하였으며 해당 체력 값을 계속 반환하여 몬스터의 체력여부를 알 수 있게 하였습니다.

```
    void MoveRandom() {
        //위 아래 좌우 랜덤 움직임
        int d = rand() % 4;
        switch (d) {
            case 0:
                if (y < 4) y++;
                break;
            case 2:
                if (y > 0) y--;
                break;
            case 1:
                if (x > 0) x--;
                break;
            case 3:
                if (x < 4) x++;
                break;
        }
    }
};
```

플레이어의 한 번의 공격이 끝난 후 몬스터가 도주하도록 설정하였습니다.
이때, 0~3 의 값을 무작위로 지니는 d 에 대한 각각의 움직임을 설정하였는데, 이또한 switch-case 문을 통해 각 d 값에 따른 움직임을 구현하였고, 제한된 지도 내에서 몬스터가 머물 수 있도록 조건문을 추가하여 지도 밖으로 벗어나지 못하게 하였습니다.

이 다음으로 플레이어의 Class 및 멤버 변수와 함수를 설명하도록 하겠습니다.

```
class Player {
public:
    int HP = 100;
    int MP = 15;
    int x = 0, y = 0;
    int P_dmg = 10;
    Player();
    Player(int x, int y){
        this -> HP = 100;
        this -> MP = 15;
        this -> x = x;
        this -> y = y;
        this -> P_dmg = 10;
    }
```

이또한 공지사항과 동일한 구성을 지니도록 하였으며, 서로의 움직임 대비 넓은 맵으로 인한 진행속도 더딤을 해결하기 위하여 플레이어의 체력을 100, MP 를 15 로 설정하였으며 초기 좌표를 (0,0), 플레이어 공격력을 10 으로 초기화하였습니다.

이후 생성자 함수와 내부 멤버 함수 값을 this ->를 통해 저장하였습니다.

```
void Show_status() {
    //상태 표시 S
    std::cout << "HP: ";
    std::cout << HP << std::endl;
    std::cout << "MP: ";
    std::cout << MP << std::endl;
    std::cout << "Position:";
    std::cout << x;
    std::cout << ",";
    std::cout << y << std::endl;
}
```

플레이어의 상태 표시에 관한 함수입니다.

std::cout 을 활용하기 위하여 iostream 을 헤더파일에서 호출하였으며 각각 변동되는 멤버 변수를 포매팅하여 출력하도록 하였습니다.

```
void X_move(int move) {
    //좌우 움직임 R, L
    if (x + move >= 0 && x + move <= 5) {
        //지도 범위 제한
        x += move;
    }
}

void Y_move(int move) {
    //위 아래 움직임
    if (y + move >= 0 && y + move <= 5) {
        //지도 범위 제한
        y += move;
    }
}
```

플레이어의 움직임과 관련된 함수입니다.

5*5 로 제한된 지도에서 벗어나지 못하도록 조건문을 구성하였으며, 현재의 좌표(x,y)에서 움직이는 값 move(+-1)를 더하는 방식으로 구현하였습니다.

```
void Attack(Monster &target) {
    if (MP <= 0) {
        std::cout << "MP 부족 !" << std::endl;
        HP = 0;
        return;
    }
    MP--;

    if (x == target.x && y == target.y) {
        //동일 위치면 공격 유효 아니면 MP 낭비
        target.Be_Attacked(P_dmg);
        std::cout << "공격 성공 ! Monster HP: ";
        std::cout << target.HP << std::endl;

        //30% 확률 반격
        if ((rand() % 100) < 30) {
            HP -= target.M_dmg;
            std::cout << "공격 받음 ! Player HP: ";
            std::cout << HP << std::endl;
        }
        //공격 이후 랜덤 움직임
        target.MoveRandom();
        //몬스터 위치 공개 코드 - 디버깅용
        std::cout << "몬스터가 (";
        std::cout << target.x;
        std::cout << ",";
        std::cout << target.y;
        std::cout << ")로 도망 !" << std::endl;

    } else {
        //동일 위치에 있지 않은 상태에서 공격 시도 -> MP만 소비
        std::cout << "공격 실패 !" << std::endl;
    }
}
```

플레이어의 공격에 관한 함수입니다.
우선적으로 현재 플레이어가 가지고 있는 MP 량을 확인하며 MP 가 0 이하일 경우 HP 를 0 으로 변경하며 게임을 종료하게 하였으며, 공격을 한 만큼 MP 를 감소시키기 위해 아래의 MP—를 작성하게 되었습니다.
공격 행위와 관련하여 현재의 5*5 의 지도는 작은 크기이므로 이를 위해 플레이어와 몬스터가 서로 같은 좌표에 존재하는지를 target.x, target.y 를 플레이어의 멤버 변수 x,y 와 비교하며 둘이 일치할 시 유효타, 아닐 시 MP 를 낭비하는 결과를 불러오도록 하였습니다.
서로가 같은 좌표에 위치한 경우 Be_Attacked()함수 동작을 한 후 몬스터의 무작위 확률의 반격과 함께 Move.Random 함수를 동작하여 몬스터가 도주를 지속하도록 하였습니다.

3. 실행 결과

(예외 처리)

```
sion@sion-Laptop:~/intern_ws/hw3/build$ ./test
Type Command(A/U/D/R/L/S): 1

Invaild Input
Type Command(A/U/D/R/L/S): f

Invaild Input
Type Command(A/U/D/R/L/S): -

Invaild Input
Type Command(A/U/D/R/L/S): AUD

공격 실패 !
Type Command(A/U/D/R/L/S):
Y Position 1 moved!
Type Command(A/U/D/R/L/S):
Y Position -1 moved!
Type Command(A/U/D/R/L/S):
```

(전체적인 진행)