소프트웨어공학과 214456 박진관

# 1. 서론

(1) 프로젝트 목적 및 배경: 7주차까지 배운 내용에 대한 실습을 위해 진행
(2) 목표: 간단한 Mud 게임 구현

# 2. 요구사항

(1) 사용자 요구사항: 유저가 상하좌우로만 이동하며 목적지에 도착하는 게임
(2) 기능계획
① 유저는 체력 20을 가지고 게임 시작
② 사용자가 이동할 때 마다 사용자 체력 1씩 감소
③ 처음 명령문을 입력 받을 때 마다 HP 함께 출력
④ HP가 0이 되면 “실패”를 출력하고 종료
⑤ 무기/갑옷, 포션, 적을 만났을 때 그에 대한 메시지를 출력

(3) 함수계획 : 반복되는 코드를 최대한 함수화로 간략화

# 3. 설계 및 구현

**맵 배열과 위치 좌표로 상태 관리**

```
int main() {
    int map[mapY][mapX] = { {0, 1, 2, 0, 4},
                            {1, 0, 0, 2, 0},
                            {0, 0, 0, 0, 0},
                            {0, 2, 3, 0, 0},
                            {3, 0, 0, 0, 2} };

    int user_x = 0; // 유저의 가로 위치
    int user_y = 0; // 유저의 세로 위치
    int health = initialHealth; // 유저의 체력
    bool hasItem = false; // 아이템 보유 여부를 나타내는 플래그
```

사용자 위치는 user_x와 user_y로 표현되며, 초기 위치는 (0, 0)으로 설정된다. 사용자는 명령어에 따라 좌표가 업데이트되고, 이 좌표가 배열 내에서 사용자의 현재 위치를 나타낸다.
프로그램은 맵 배열과 사용자의 위치 좌표를 통해 이동과 상호작용을 처리한다.

**명령어 입력과 이동 처리**

```
while (true) {
    string user_input = "";
    cout << "명령어를 입력하세요 (up, down, left, right, map, exit): ";
    cin >> user_input;

    if (user_input == "up") {
        movePlayer(user_x, user_y, 0, -1, map, health, hasItem);
    } else if (user_input == "down") {
        movePlayer(user_x, user_y, 0, 1, map, health, hasItem);
    } else if (user_input == "left") {
        movePlayer(user_x, user_y, -1, 0, map, health, hasItem);
    } else if (user_input == "right") {
        movePlayer(user_x, user_y, 1, 0, map, health, hasItem);
    } else if (user_input == "map") {
        displayMap(map, user_x, user_y);
    } else if (user_input == "exit") {
        cout << "게임을 종료합니다." << endl;
        break;
    } else {
        cout << "잘못된 입력입니다. up, down, left, right, map, exit 중에서 선택하세요." << endl;
        continue;
    }
```

프로그램은 cin을 사용해 사용자 명령어를 입력받음. 명령어가 입력되면 if 문을 통해 명령어를 해석하여 이동 방향을 결정.
예를 들어 up 명령어가 입력되면 movePlayer 함수가 호출되고, 해당 함수 내에서 dx와 dy 값을 통해 새로운 좌표가 계산됨.

**좌표 기반 이동 처리와 경계 체크**

```
// 이동하려는 위치가 유효한지 확인 후 이동하는 함수
void movePlayer(int &user_x, int &user_y, int dx, int dy, int map[][mapX], int &health, bool &hasItem) {
    int new_x = user_x + dx;
    int new_y = user_y + dy;
    if (!checkXY(new_x, mapX, new_y, mapY)) {
        cout << "맵을 벗어났습니다. 다시 돌아갑니다." << endl;
    } else {
        user_x = new_x;
        user_y = new_y;
        cout << (dy == -1 ? "위" : dy == 1 ? "아래" : dx == -1 ? "왼쪽" : "오른쪽") << "으로 한 칸 이동합니다." << endl;
        health--; // 이동 시 체력 감소
        checkEncounter(map, user_x, user_y, health, hasItem); // 만남 이벤트 확인
        displayMap(map, user_x, user_y);
    }
}
```

movePlayer 함수는 이동 방향을 나타내는 dx와 dy를 사용하여 새로운 좌표(new_x, new_y)를 계산한다. 새로운 좌표가 맵의 경계를 벗어나지 않도록 checkXY 함수를 통해 확인한다. 경계를 벗어날 경우, 이동이 취소되고 경고 메시지가 출력.
경계를 벗어나지 않는다면 새로운 위치로 이동하며 체력이 1 감소. 이후 해당 위치에서 이벤트가 발생할 수 있으므로 checkEncounter 함수를 통해 만남 이벤트를 처리함.

**맵 상호작용 원리**

```
// 아이템, 포션, 적을 만났을 때의 처리를 담당하는 함수
void checkEncounter(int map[][mapX], int user_x, int user_y, int &health, bool &hasItem) {
    int posState = map[user_y][user_x];
    switch (posState) {
        case 1: // 아이템
            cout << "아이템을 발견했습니다! 다음 적과의 전투에서 피해를 입지 않습니다." << endl;
            hasItem = true; // 아이템 획득 시 다음 적 피해 방지
            map[user_y][user_x] = 0; // 아이템을 발견한 후 빈 공간으로 변경
            break;
        case 2: // 적
            if (hasItem) {
                cout << "적을 만났지만, 아이템 덕분에 피해를 입지 않았습니다!" << endl;
                hasItem = false; // 아이템 사용 후 초기화
            } else {
                cout << "적을 만났습니다! 체력이 2 감소했습니다." << endl;
                health -= 2; // 아이템이 없을 경우에만 피해
            }
            // 적은 위치에 그대로 남아있음
            break;
        case 3: // 포션
            cout << "포션을 발견하여 체력이 5 증가했습니다." << endl;
            health += 5;
            // 포션은 위치에 그대로 남아있음
            break;
    }
}
```

checkEncounter 함수는 사용자가 이동한 위치의 상태(posState)를 확인하여 이벤트 실행. 각 위치의 상태에 따라 다른 이벤트가 발생함.

아이템 (1): hasItem 변수를 true로 설정해 다음 적과의 만남에서 피해를 방지. 아이템은 일회성으로 사용되므로 맵에서 0으로 변경.

적 (2): 사용자가 아이템을 가지고 있지 않으면 체력이 2 감소. 아이템이 있을 경우 피해를 받지 않고 아이템이 소멸.

포션 (3): 체력을 5만큼 회복. 포션은 계속 맵에 남아있어, 포션 위치를 지날 때마다 체력이 회복함.

**맵 출력 및 사용자 위치 시각화**

```
// 지도와 사용자 위치 출력하는 함수
void displayMap(int map[][mapX], int user_x, int user_y) {
    for (int i = 0; i < mapY; i++) {
        for (int j = 0; j < mapX; j++) {
            if (i == user_y && j == user_x) {
                cout << " USER |";
            } else {
                int posState = map[i][j];
                switch (posState) {
                case 0:
                    cout << "      |";
                    break;
                case 1:
                    cout << "아이템|";
                    break;
                case 2:
                    cout << "  적  |";
                    break;
                case 3:
                    cout << " 포션 |";
                    break;
                case 4:
                    cout << "목적지|";
                    break;
                }
            }
        }
        cout << endl;
        cout << " ------------------------------- " << endl;
    }
}
```

displayMap 함수는 현재 맵과 사용자 위치를 콘솔에 출력.
USER라는 텍스트로 사용자 위치를 시각적으로 표시하여 현재 캐릭터가 어디에 위치하고 있는지 쉽게 확인이 가능함.
각 칸에 대한 상태(0, 1, 2, 3, 4)를 조건에 맞게 텍스트로 변환하여 출력함으로써 맵의 상태를 한눈에 볼 수 있음.

**게임 종료 조건 확인**

```
// 체력 확인
cout << "현재 체력: " << health << endl;
if (health <= 0) {
    cout << "체력이 모두 소진되었습니다. 게임 오버!" << endl;
    break;
}

// 목적지 도착 확인
if (checkGoal(map, user_x, user_y)) {
    cout << "목적지에 도착했습니다! 축하합니다!" << endl;
    cout << "게임을 종료합니다." << endl;
    break;
}
```

게임은 사용자의 체력이 0 이하가 되거나, 목적지(4)에 도착할 때 종료되는 것으로 설정함.
체력이 0 이하로 떨어지면 게임 오버 메시지가 출력되고, checkGoal 함수가 true를 반환할 경우 사용자가 목적지에 도착한 것으로 간주하여 축하 메시지를 출력함.

**작동 원리 요약**

**(1) 좌표와 배열을 활용한 위치 및 상태 관리:** 맵은 2차원 배열로 구성되어 있고, 각 위치에 있는 요소는 배열의 값으로 표현함. 사용자의 위치는 좌표로 관리되며, 이동할 때마다 좌표 값이 업데이트됨.

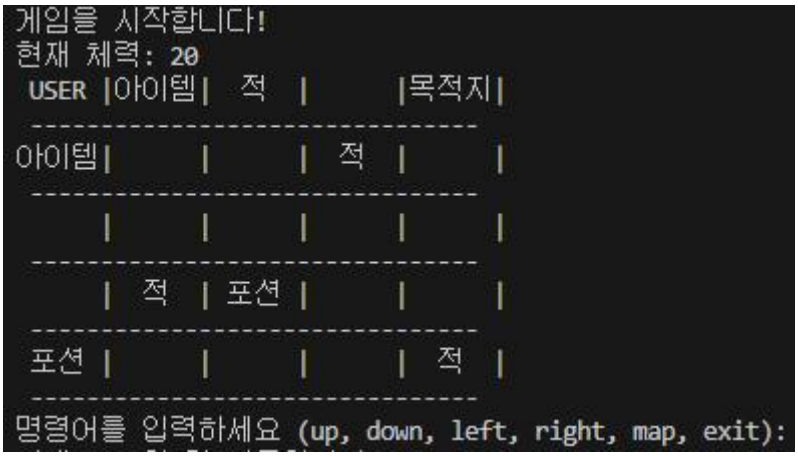
**(2) 이동 및 이벤트 처리:** 사용자가 이동할 때마다 movePlayer와 checkEncounter 함수가 호출되어 이동한 위치에서 이벤트가 발생함. 이로 인해 체력 소모, 아이템 획득, 적과의 전투 등의 상호작용이 발생.

**(3)맵 경계 및 목표 확인:** 맵의 경계를 벗어나지 않도록 checkXY 함수가 경계 확인을 수행하며, 목표 지점에 도달했는지 checkGoal 함수로 확인하여 게임 종료 조건을 관리함.
이러한 방식으로 프로그램은 입력 명령에 따라 사용자의 위치와 상태를 실시간으로 업데이트하며, 게임의 진행 상황을 반영한다.

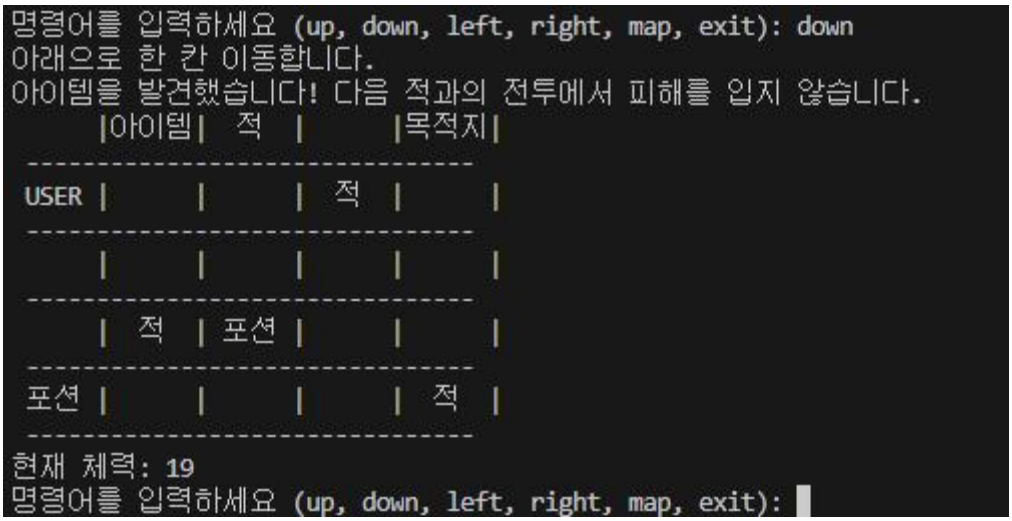
소프트웨어공학과 214456 박진관

**각 기능 테스트 결과**

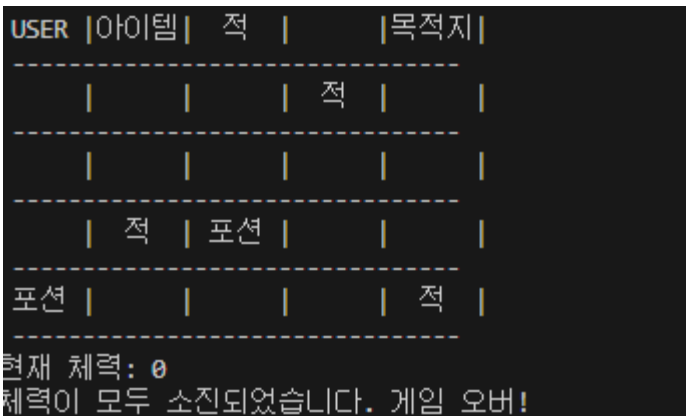
① 유저가 상하좌우로만 이동하며 목적지에 도착하는 게임, 유저는 체력 20을 가지고 게임 시작, 이것을 초기 상태 출력.



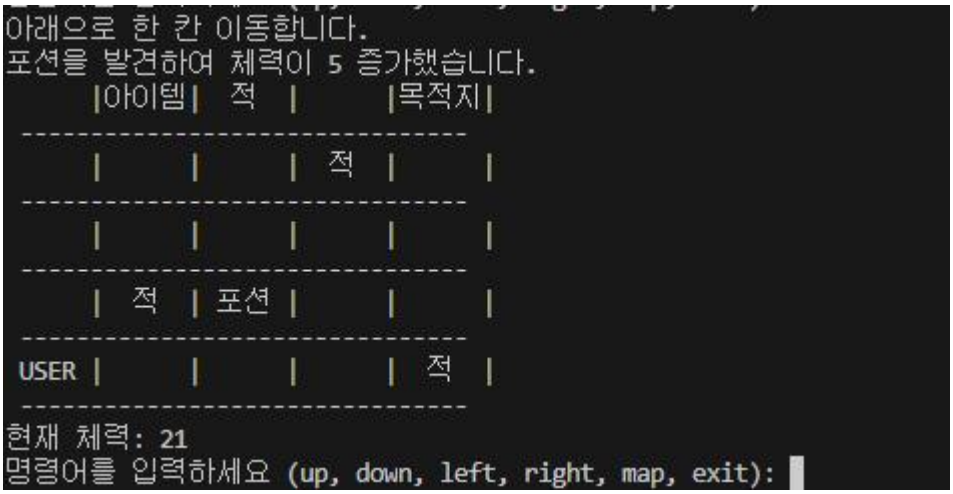

② 사용자가 이동할 때 마다 사용자 체력 1씩 감소, ③ 처음 명령문을 입력 받을 때 마다 HP 함께 출력



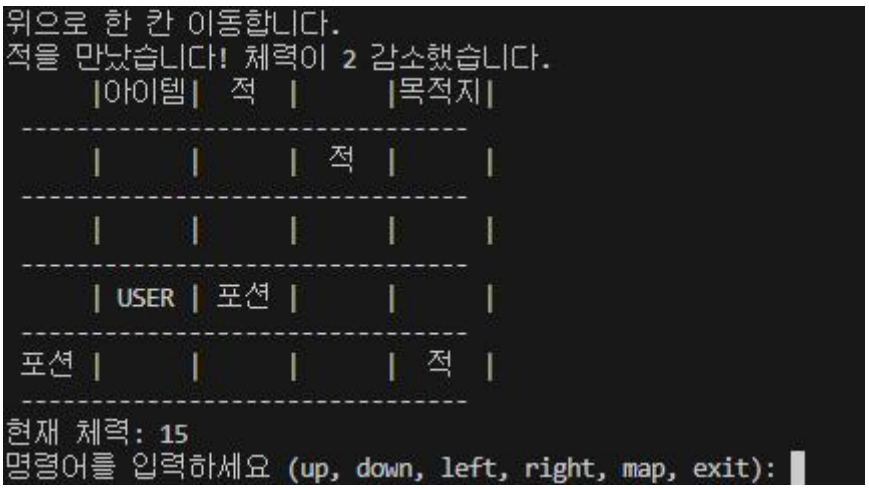

④ HP가 0이 되면 “실패”를 출력하고 종료



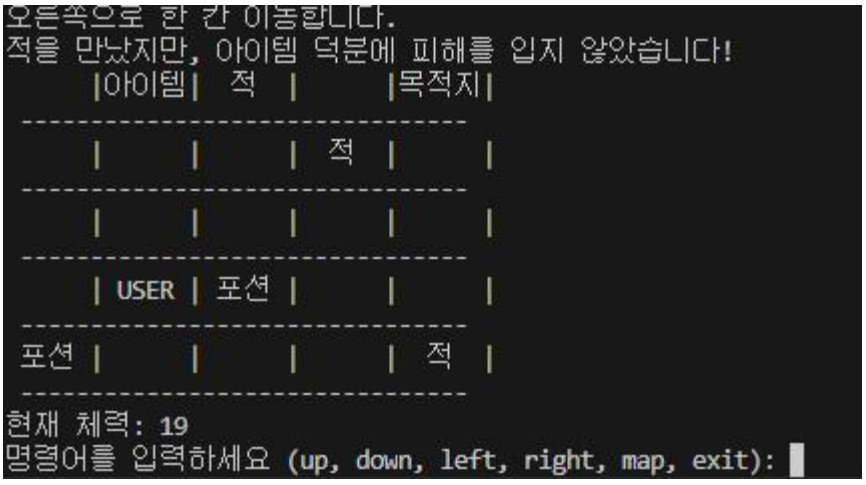
⑤ 무기/갑옷, 포션, 적을 만났을 때 그에 대한 메시지를 출력
(포션 발견)



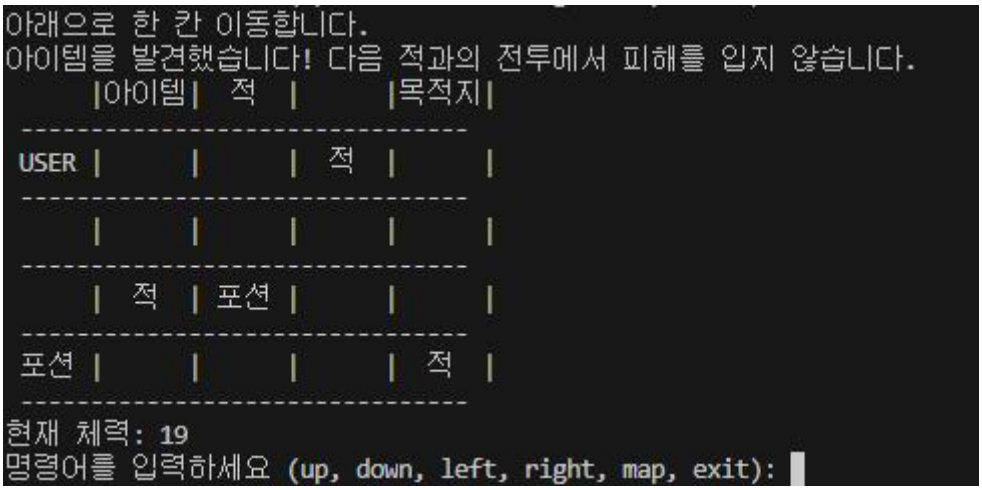

(적 발견)



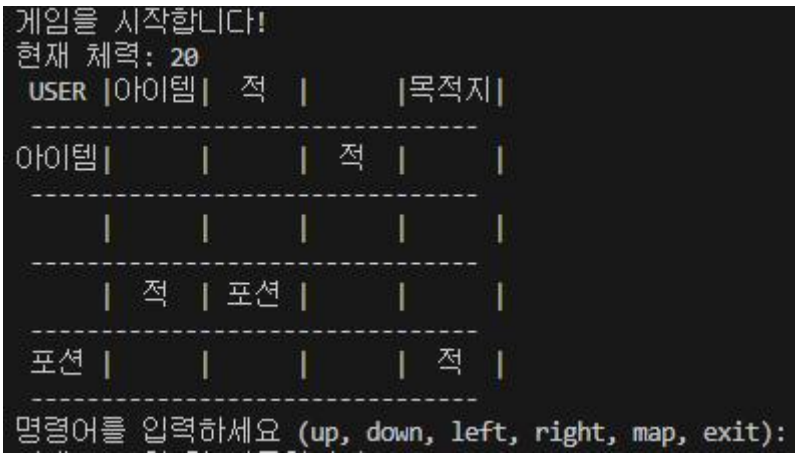

(아이템을 가졌을 때 적 발견)

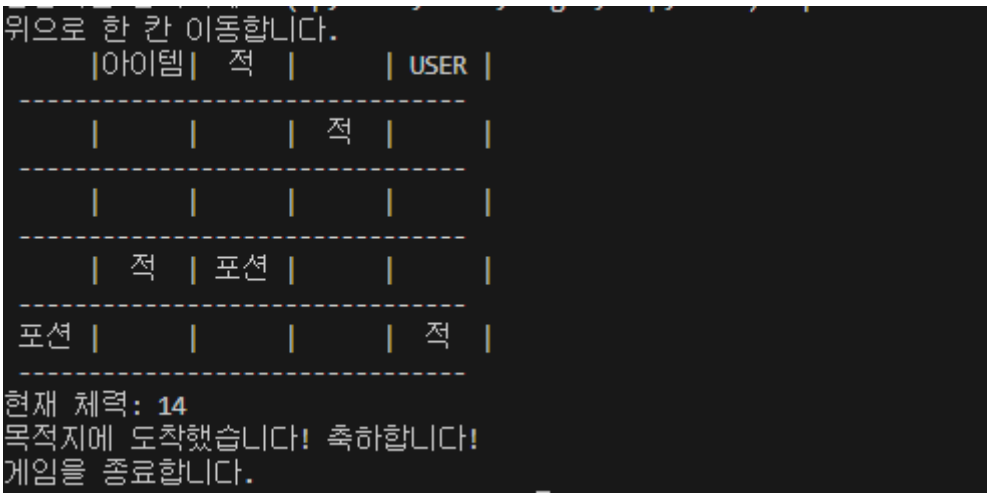


(아이템 발견)




## 4. 테스트




위 추가 기능에 맞게 mud 게임이 잘 작동되는것이 확인되었다.

# 5. 결과 및 결론

최종적으로 mud프로그램이 잘 작동하였으나, 과정이 순탄치 않았다.

첫 번째는 프로그램 테스트를 할려고 명령어를 입력하는데 계속 잘못된 입력이라고 뜨는 오류가 있었다. 조건문에도 전혀 오류가 없고, 코드가 꼬이지도 않았는데 잘못된 입력만 반복되었는데, 결국 이건 입력을 한글로 받아서 오류가 생긴것이었다.(...)
그래서 입력은 영어로 수정하였고, 대신 출력은 한글로 계속 유지되게 하였다.

두 번째는 moveplayer 함수를 만들겠다고 했는데, 이걸 순서를 잘못되게 하였다. 아이템 시스템을 추가한 후에 만들었어야 했는데, moveplayer 함수를 만든 후 아이템 시스템을 추가해버려서 아이템이 계속 남아있는 버그라던가, 아이템을 먹고 적을 만났는데도 피해를 계속 받는 버그가 생겨서 미쳐버리는 줄 알았다. 결국 이 버그는 hasitem 변수를 추가하여 해결하였다.