< 머드게임 보고서 >

185482 김기태

1. 서론

- 목적 및 배경 : 실습한 부분을 활용하기 위하여 제작하였다.
- 목표 : 어떻게든 돌아가는 머드게임을 구현하는 것이다.

2. 요구사항

- 사용자 요구사항 :
    1) 상하좌우로 유저가 움직인다.
    2) 아이템, 포션, 적을 마주칠 때마다 상호작용이 존재해야한다.
    3) 지도를 매 턴마다, 혹은 원할 시 출력할 수 있어야 한다.
    4) 매 턴마다 체력을 표기해야 한다.
    5) 사용자 이동 시마다 체력을 1씩 감소시켜야 한다.
    6) 유저의 시작 체력은 20이다.

- 기능 계획 :
    1) 맵을 출력한다.
    2) 각 맵에 유저와 아이템, 적, 포션, 골인지점 등을 배치한다.
    3) 명령어를 쳐서 움직이도록 한다.
    4) 각 상황에 맞게 체력이 증감되고 이것이 매 턴마다 표시되도록 한다.

- 함수 계획 :
    1) 맵 정의 함수(displayMap)
    2) 좌표 유효성 검사 함수(checkXY)
    3) 목표달성 확인 함수(checkGoal)
    4) 오브젝트 상호작용 함수(checkState)
    5) 체력 관리 함수(healthCare)

3. 설계 및 구현

< 상수, 변수, 함수 선언부분 >

```
// 상수 선언
const int mapX = 5;
const int mapY = 5;
const int default_health = 20;

// 변수 선언
bool ongoing = true; // 이 bool은 게임이 진행중인지 여부를 확인. 기본값 true.
bool debug_mod = false; // 이 bool은 디버그 모드 상태를 확인. 기본값 false.
int health = default_health; // 현재 체력 설정.

// 사용자 정의 함수
bool checkXY(int user_x, int mapX, int user_y, int mapY);
void displayMap(int map[][mapX], int user_x, int user_y);
bool checkGoal(int map[][mapX], int user_x, int user_y);
void checkState(int map[][mapX], int user_x, int user_y, int& health);
void debugMod(int map[][mapX], int user_x, int user_y);
void healthCare(int& health);
```

▣ 입력

mapX, mapY: int로 선언된 지도의 가로와 세로 크기를 나타내는 상수이다.
default_health: int로 선언된 게임 시작 시 플레이어의 기본 체력을 주는 상수이다. 20으로 초기화되어 있다.
ongoing: bool로 선언된 게임 진행 상태를 나타내는 변수이다. 기본 값은 true이다.
debug_mod: bool로 선언된 디버그 모드의 상태를 확인하는 변수이다. 기본 값은 false이다.
health: int로 선언된 현재 플레이어의 체력을 나타내는 변수이다. 기본 값은 default_health로 초기화되어 있다.

▣ 함수

bool checkXY(int user_x, int mapX, int user_y, int mapY): 사용자의 위치가 지도 범위 내에 있는지 검사한다.
void displayMap(int map[][mapX], int user_x, int user_y): 지도와 사용자의 위치를 출력한다.
bool checkGoal(int map[][mapX], int user_x, int user_y): 사용자가 목적지에 도달했는지 여부를 확인한다.
void checkState(int map[][mapX], int user_x, int user_y, int& health): 현재 위치 확인 후 이벤트를 발생한다.
void debugMod(int map[][mapX], int user_x, int user_y): 디버그 모드일 때 추가 정보를 출력한다.
void healthCare(int& health): 체력 관리 및 사망 여부를 확인한다.

▣ 설명

상수와 변수, 함수를 선언한다.

< 메인 함수 >

```
// 메인 함수
int main() {
    // 0은 빈 공간, 1은 아이템, 2는 적, 3은 포션, 4는 목적지
    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} };


    // 유저의 위치를 저장할 변수
    int user_x = 0; // 가로 번호
    int user_y = 0; // 세로 번호

    // 게임 시작
    while (ongoing == true) { // 사용자에게 계속 입력받기 위해 무한 루프

        // 사용자의 입력을 저장할 변수
        string user_input = "";

        if (debug_mod == true)
        {
            debugMod(map, user_x, user_y);
        }
        cout << "HP = " << health << endl;
        cout << "명령어를 입력하세요 (상,하,좌,우,지도,종료,디버그): ";
```

```
cin >> user_input;

if (user_input == "상") {
    // 위로 한 칸 올라가기
    user_y -= 1;
    bool inMap = checkXY(user_x, mapX, user_y, mapY);
    if (inMap == false) {
        cout << "맵을 벗어났습니다. 다시 돌아갑니다." << endl;
        user_y += 1;
    }
    else {
        cout << "위로 한 칸 올라갑니다." << endl;
        displayMap(map, user_x, user_y);
        checkState(map, user_x, user_y, health);
        health -= 1;
    }
}
else if (user_input == "하") {
    // TODO: 아래로 한 칸 내려가기
    user_y += 1;
    bool inMap = checkXY(user_x, mapX, user_y, mapY);
    if (inMap == false) {
        cout << "맵을 벗어났습니다. 다시 돌아갑니다." << endl;
        user_y -= 1;
    }
    else {
```

```
        cout << "위로 한 칸 내려갑니다." << endl;
        displayMap(map, user_x, user_y);
        checkState(map, user_x, user_y, health);
        health -= 1;
    }
}
else if (user_input == "좌") {
    // TODO: 왼쪽으로 이동하기
    user_x -= 1;
    bool inMap = checkXY(user_x, mapX, user_y, mapY);

    if (inMap == false) {
        cout << "맵을 벗어났습니다. 다시 돌아갑니다." << endl;
        user_x += 1;
    }
    else {
        cout << "왼쪽으로 이동합니다." << endl;
        displayMap(map, user_x, user_y);
        checkState(map, user_x, user_y, health);
        health -= 1;
    }
}
else if (user_input == "우") {
    // TODO: 오른쪽으로 이동하기
    user_x += 1;
    bool inMap = checkXY(user_x, mapX, user_y, mapY);
```

```
        if (inMap == false) {
            cout << "맵을 벗어났습니다. 다시 돌아갑니다." << endl;
            user_x -= 1;
        }
        else {
            cout << "오른쪽으로 이동합니다." << endl;
            displayMap(map, user_x, user_y);
            checkState(map, user_x, user_y, health);
            health -= 1;
        }
    }
    else if (user_input == "지도") {
        // TODO: 지도 보여주기 함수 호출
        displayMap(map, user_x, user_y);
    }
    else if (user_input == "종료") {
        cout << "종료합니다.";
        break;
    }
    else if (user_input == "디버그" && debug_mod == false)
    {
        cout << "디버그 모드 on" << endl;
        debug_mod = true;
    }
    else if (user_input == "디버그" && debug_mod == true)
    {
        cout << "디버그 모드 off" << endl;
        debug_mod = false;
    }
    else {
        cout << "잘못된 입력입니다." << endl;
        continue;
    }

    // 목적지에 도달했는지 체크
    bool finish = checkGoal(map, user_x, user_y);
    if (finish == true) {
        cout << "목적지에 도착했습니다! 축하합니다!" << endl;
        cout << "게임을 종료합니다." << endl;
        break;
    }
    healthCare(health); // 헬스케어 부분은 체력이 0으로 떨어졌는지 확인하기 위한 목적
}
return 0;
}
```

▣ 입력

map : 게임 지도의 상태, 아이템, 적, 포션, 목적지의 위치 정보를 포함하는 2차원 배열이다.
user_x, user_y : 사용자의 위치를 나타내는 변수이다.
health : 사용자의 체력을 나타내는 변수이다.
user_input : 사용자의 입력을 저장하는 문자열이다.

▣ 처리

while 루프를 사용하여 ongoing이 true라면 계속 루프가 실행된다.
사용자로부터 명령어 입력을 받는다. (상, 하, 좌, 우, 지도, 종료, 디버그)
사용자의 입력에 따라 지도에서의 위치 이동, 상태 변화, 목적지 도달 여부까지 확인한다.
각 이동 후 사용자의 위치, 지도 상태, 체력 감소 등을 처리한다.
디버그 모드 활성화시 숨겨진 추가 정보가 제공된다.

▣ 출력

지도 상태, 지도 속 사용자 위치, 현재 체력 등이 계속 출력된다.
출력 문구로는 사용자 입력에 따른 상황 변화 및 메시지, 게임 종료 메시지 등이 있다.

▣ 설명

게임은 사용자가 '종료' 명령을 입력하거나, Goal조건이 맞춰질 때 break문으로 탈출하게 된다.
사용자는 상, 하, 좌, 우 방향으로 이동할 수 있으며, 이동 시 마다 체력이 1씩 감소한다.
맵에서 벗어나는 이동을 시도하면 경고 메시지를 출력하고 이동을 취소한다.
checkState 함수를 통해 이동 후의 상태(적과의 전투, 아이템 및 포션의 획득)를 확인한다.
healthCare 함수는 체력 상태를 관리하고, 체력이 0이 되면 게임이 종료될 수 있다.

< 지도출력 함수 >

```
// 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 |"; // 양 옆 1칸 공백
            }
            else {
                int posState = map[i][j];
                switch (posState) {
                case 0:
                    cout << "      |"; // 6칸 공백
                    break;
                case 1:
                    cout << "아이템|";
                    break;
                case 2:
                    cout << "  적  |"; // 양 옆 2칸 공백
                    break;
                case 3:
                    cout << " 포션 |"; // 양 옆 1칸 공백
                    break;
```

```
                case 4:
                    cout << "목적지|";
                    break;
                }
            }
        }
        cout << endl;
        cout << " ------------------------------ " << endl;
    }
}
```

▣ 입력

map[][] : 지도(2차원 배열)

user_x, user_y : 사용자의 현재 위치 좌표

▣ 반환값

void함수이기에 없다.

▣ 결과

게임 맵 전체와 사용자의 현재 위치를 출력한다.

▣ 설명

이 함수는 2차원 맵과 사용자의 위치, 아이템과 적과 같은 기타 오브젝트의 위치, 목적지의 위치를 표시한다.
사용자의 위치는 'USER'로 표시된다.

< 좌표 유효성 검사 함수 >

```
// 이동하려는 곳이 유효한 좌표인지 체크하는 함수
bool checkXY(int user_x, int mapX, int user_y, int mapY) {
    bool checkFlag = false;
    if (user_x >= 0 && user_x < mapX && user_y >= 0 && user_y < mapY) {
        checkFlag = true;
    }
    return checkFlag;
}
```

▣ 입력

user_x, user_y : 사용자의 현재 위치 좌표(여기서는 이동하고자 하는 위치)

mapX, mapY: 맵의 가로 및 세로 크기

▣ 반환값
checkFlag: 이동하려는 좌표가 맵 내에 있는지 여부를 나타내는 bool (true or false)

▣ 결과
사용자가 이동하려는 좌표가 맵 내에 존재하는지 확인 후, 아닐 경우 원래 위치로 돌아오고 루프를 다시 시작한다.

▣ 설명
이 함수는 사용자가 이동하려는 좌표가 맵의 범위 내에 있는지 확인한다.
조건을 만족할 경우 checkFlag를 true로 설정하고, 그렇지 않을 경우 false를 반환한다.
사용자가 이동하려는 좌표가 정상적이지 않은 경우, 이동 불가 문구를 표기하고 원래 위치로 복귀한다.

< 골체크 함수 >

```
// 유저의 위치가 목적지인지 체크하는 함수
bool checkGoal(int map[][mapX], int user_x, int user_y) {
    // 목적지 도착하면
    if (map[user_y][user_x] == 4) {
        return true;
    }
    return false;
}
```

▣ 입력
map[][] : 지도(2차원 배열)
user_x, user_y : 사용자의 현재 위치 좌표

▣ 반환값
bool값 true or false.

▣ 결과
사용자의 현재 위치가 목적지면 루프를 빠져나온다.

▣ 설명
사용자의 현재 위치가 목적지(map[user_y][user_x] == 4)라면 true를 반환하여 목적지 도달을 알리고 메인 함수에서 break문으로 루프를 빠져나오도록 한다. 그렇지 않다면 false를 반환하며, 이 경우 루프가 계속된다.

< 위치 체크 함수 >

```
void checkState(int map[][mapX], int user_x, int user_y, int& health)
{
    // 아이템 먹으면 반응
    if (map[user_y][user_x] == 1)
    {
        cout << "아이템을 먹었습니다." << endl;
    }
    // 적과 마주치면 HP -2
    else if (map[user_y][user_x] == 2)
    {
        cout << "적과 마주쳤습니다." << endl;
        cout << "체력을 2만큼 잃습니다." << endl;
        health -= 2;
    }
    // 포션을 먹으면 HP +2
    else if (map[user_y][user_x] == 3)
    {
        cout << "포션을 먹고 체력을 2만큼 회복합니다." << endl;
        health += 2;
    }
    else
    {
    }
}
```

▣ 입력

map[][] : 지도(2차원 배열)

user_x, user_y : 사용자의 현재 위치 좌표

health: 사용자의 현재 체력 (참조로 함수 바깥의 값이 수정된다)

▣ 반환값

void함수이기에 없다.

▣ 결과

사용자가 아이템, 적, 포션과 같은 좌표에 위치할 시 경우에 따라 적절한 출력과 함께 health 변수의 값을 변경한다.

▣ 설명

아이템 (1): 아이템을 먹었다는 메시지를 출력한다.

적 (2): 적과 마주쳤다는 메시지를 출력하고, 체력은 2만큼 감소한다.

포션 (3): 포션을 먹었다는 메시지를 출력하고, 체력은 2만큼 증가한다.

health 파라미터는 참조로 전달되므로, 함수 내에서 health 값을 변경하면 메인 함수에서도 값이 바뀐다.

< 체력 함수 >

```
// 체력 관련 부분
void healthCare(int& health)
{
    // 최대 체력 20 제한
    if (health > 20)
    {
        health = default_health;
    }
    // 체력이 0이 될 시 게임 오버
    else if (health <= 0)
    {
        cout << "HP가 바닥났기에 실패했습니다." << endl;
        cout << "게임을 종료합니다," << endl;
        ongoing = false;
    }
}
```

▣ 입력

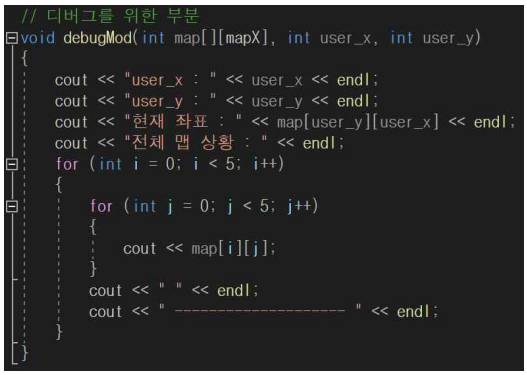
health: 사용자의 현재 체력 (참조로 함수 바깥의 값이 수정된다)

▣ 반환값
void함수이기에 없다.

▣ 결과
만일 20을 넘는 회복이 이루어지면 다시 20으로 강제 재조정된다.
만일 체력이 0이하로 떨어지면 실패 문구가 출력되며 ongoing은 false가 된다.

▣ 설명
체력이 20을 초과하는 경우, 체력을 20으로 강제 조정한다. 이는 최대 체력을 조정하기 위한 것이다.
체력이 0 이하가 되는 경우, 사용자에게 실패 문구를 출력하고 ongoing을 false로 설정하여 while루프의 조건을 무력화하여 루프를 빠져나가서 게임을 종료시킨다.

< 디버그 모드 함수 >



```
// 디버그를 위한 부분
void debugMod(int map[][mapX], int user_x, int user_y)
{
    cout << "user_x : " << user_x << endl;
    cout << "user_y : " << user_y << endl;
    cout << "현재 좌표 : " << map[user_y][user_x] << endl;
    cout << "전체 맵 상황 : " << endl;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            cout << map[i][j];
        }
        cout << " " << endl;
        cout << " -------------------- " << endl;
    }
}
```

▣ 입력
map[][] : 지도(2차원 배열)
user_x, user_y : 사용자의 현재 위치 좌표

▣ 반환값
void함수이기에 없다.

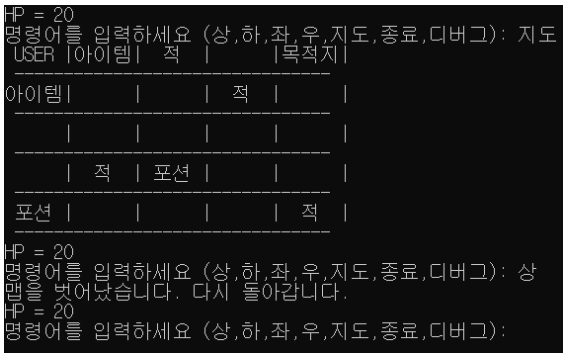
▣ 결과
사용자의 현재 위치와 맵의 상태를 알 수 있는 숫자들을 콘솔에 출력한다.

▣ 설명
debugMod 함수는 디버깅을 위한 정보를 출력하는 부분이다.

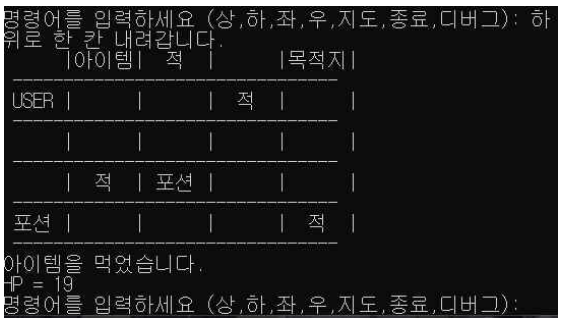
4. 테스트

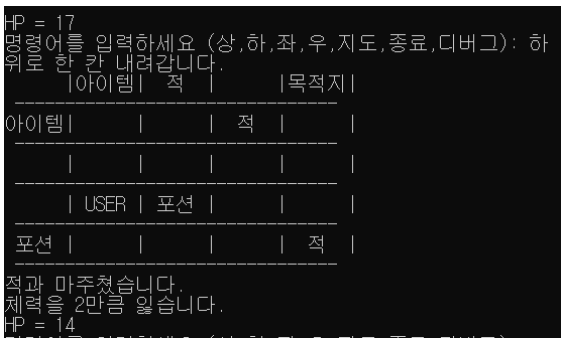
< 기능별 테스트 결과 >

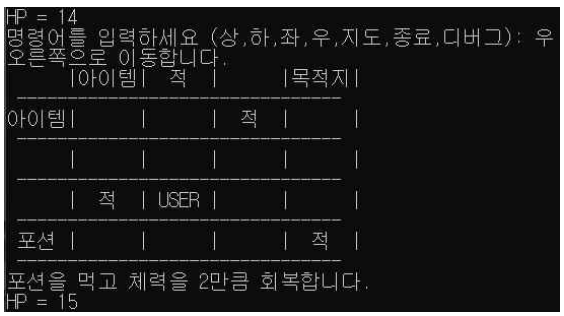
- 유효성검사 & 지도출력 (checkXY & displayMap)



- 아이템, 적, 포션과 마주쳤을 때(checkState)







- 성공(checkGoal) & 실패(healthCare)

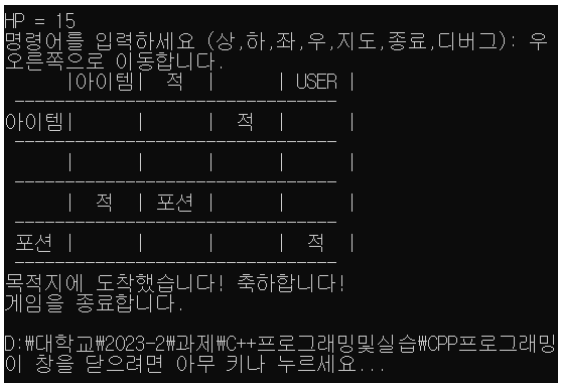
```
명령어를 입력하세요 (상,하,좌,우,지도,종료,디버그): 우
오른쪽으로 이동합니다.
      |아이템|  적  |      | USER |
 ---------------------------------
아이템|      |      |  적  |      |
 ---------------------------------
      |      |      |      |      |
 ---------------------------------
      |  적  | 포션 |      |      |
 ---------------------------------
 포션 |      |      |      |  적  |
 ---------------------------------
목적지에 도착했습니다! 축하합니다!
게임을 종료합니다.
```

```
적과 마주쳤습니다.
체력을 2만큼 잃습니다.
HP가 바닥났기에 실패했습니다.
게임을 종료합니다.

D:₩대학교₩2023-2₩과제₩C++프로
가) 종료되었습니다(코드: 0개)
```

< 전체 테스트 결과 >

```
HP = 20
명령어를 입력하세요 (상,하,좌,우,지도,종료,디버그): 우
오른쪽으로 이동합니다.
      | USER |  적  |      |목적지|
 ---------------------------------
아이템|      |      |  적  |      |
 ---------------------------------
      |      |      |      |      |
 ---------------------------------
      |  적  | 포션 |      |      |
 ---------------------------------
 포션 |      |      |      |  적  |
 ---------------------------------
아이템을 먹었습니다.
HP = 19
명령어를 입력하세요 (상,하,좌,우,지도,종료,디버그): 우
오른쪽으로 이동합니다.
      |아이템| USER |      |목적지|
 ---------------------------------
아이템|      |      |  적  |      |
 ---------------------------------
      |      |      |      |      |
 ---------------------------------
      |  적  | 포션 |      |      |
 ---------------------------------
 포션 |      |      |      |  적  |
 ---------------------------------
적과 마주쳤습니다.
체력을 2만큼 잃습니다.
HP = 16
명령어를 입력하세요 (상,하,좌,우,지도,종료,디버그): 우
오른쪽으로 이동합니다.
      |아이템|  적  | USER |목적지|
 ---------------------------------
아이템|      |      |  적  |      |
 ---------------------------------
      |      |      |      |      |
 ---------------------------------
      |  적  | 포션 |      |      |
 ---------------------------------
 포션 |      |      |      |  적  |
 ---------------------------------
```



5. 결과 및 결론

프로젝트 결과 : 어떻게든 동작하는 머드 게임을 만들었다.
느낀 점 :

1) 이런저런 기능들을 추가할 수 있을 것 같아 보인다. 하지만 제한사항이 있다 보니 완전히 내 마음대로 기능들을 짜지는 못했다. 경험치와 MP, SP, AP등도 잘만 하면 구현할 수 있어 보인다. 특히 경험치는 일정 수치에 올라가면 if else문으로 구현할 수 있어 보인다. 의사 코드로 대강이나마 적어보자면

```
if (보유 경험치 >= 레벨업 필요 경험치수 && 현재 레벨 <= 10)
{
	보유 경험치 = 0
	레벨업 필요 경험치수 * 1.5
}
else if (보유 경험치 >= 레벨업 필요 경험치수 && 현재 레벨 <= 20)
	경험치 = 0
	레벨업 필요 경험치수 * 2
}
else
{
	경험치 = 0
	레벨업 필요 경험치수 * 2.5
}
```

이렇게 표현할 수 있지 않을까 싶다.

2) 이런 것들을 잘 응용하면 맵을 점령하는 형태의 게임도 만들 수 있을 것이라고 생각된다. 점령한 지역에는 플레이어의 표식이 표기되고, if else문을 통해서 아군의 군사가 더 많으면 이기고 적군의 군사가 많으면 지고 병력수만 줄어드는 방식이다. 조금 더 응용하여, 2차원 배열을 여러개 생성하여 어떤 좌표에 해당하는 어떤 2차원 배열에는 지도의 지형에 따라 공방 보너스(ex. 산지에서 전투를 벌이면 방어측 병력수 보정 20%)를 주거나, 또는 특정한 지형에는 아예 유효성 검사에 걸려서 진입하지 못하도록 할 수도 있다. 그래픽만 없을 뿐이지, 어느 정도는 구현 가능할 것으로 보인다.