C++프로그래밍 및 실습

C++로 구현한 Rouge

진척 보고서 #3

제출일자: 2024/12/15
제출자명: 손유채
제출자 학번: 214940

1. 프로젝트 목표

1) 배경 및 필요성

다양한 게임을 접하며 자연스럽게 프로그래밍에 관심을 가지게 되었습니다. 게임을 즐길수록 저만의 게임을 만들어 세상에 선보이고 싶다는 목표가 생겼습니다. 특히 여러 장르 중에서도 매 플레이 마다 새로운 경험을 하게 하는 Roguelike 게임을 만들고 싶었습니다.

Roguelike의 시초라 할 수 있는 "Rogue"라는 게임이 있는데 이를 제 색깔로 재해석해 보고싶어 저만의 개성을 담은 Roguelike게임을 구현해 보고자 이 프로젝트의 주제로 잡았습니다.

2) 프로젝트 목표

로그의 특징인 무작위성 맵, 턴 기반 전투방식, 플레이어의 지속적인 죽음의 특징을 게임에 녹여내어 구현하는 것이 목표입니다

3) 차별점

기존의 "로그"는 여러층을 내려가는 방식이지만 이는 게임을 길게 하기에 1층만 나오게하되 매 플레이마다 맵을 바뀌게 할 예정입니다.

2. 기능 계획

1) 무작위 맵 생성

~~맵을 여러조각으로 만들어놓고 매 플레이 할때마다 맵 조각을 합쳐서 매 플레이마다 새로운 맵을 만드는 방식으로 무작위 맵을 만든다.~~

- 특정좌표에서 맵 시작점을 정해 랜덤 난수를 생성하여 시작위치 및 맵의 넓이를 정하여 무작위 맵을 만든다.

(1) 맵 생성

(2) 맵 구현
~~매 플레이시 랜덤하게 맵 조각을 불러와 연결하여 맵을 만듦~~

2) 플레이어 및 적들의 이동 구현

- 방향키 입력을 받아 플레이어의 이동을 구현하고 이를 1턴으로 잡아 적들이 각 성향에 따라 이동하게 구현하게 한다

(1) 플레이어 이동 구현

(2) 적 이동 구현

3) 매 시도로 나오는 변화

- 플레이어의 죽음이나 목표달성으로 아이템의 해금, 맵의 변화, 플레이 난이도를 조정.

4) 시작화면, 시작장소 구현

- 게임 시작 시 바로 시작하는게 아닌 시작장소와 시작화면을 구현하여 게임에 대한 설명이나 시작 상태를 알 수 있게 해줌.

(1) 시작화면 만들기

(2) 시작장소 만들기

5) 다양한 옵션 구현

- 특정키 입력 시 일시정지, 플레이어 상태 등 게임에 필요한 옵션을 구현.

(1) 일시정지 구현

(2) 플레이어 상태 확인

3. 진척사항

1) 기능 구현 – 최대한 헤더파일을 이용하려 했음

(1) 맵 생성

```
class Position
{
public:
    int x, y; // 각 개체의 x,y좌표
};

class Door
{
public:
    Position coordinate; // 장소에 있는 문의 좌표
    int connect;         // 문 연결여부
};

class Map
{
public:
    Position coordinate; // 장소가 시작되는 x,y좌표
    int height, width;   // 장소의 가로,세로길이
    Door *doors;         // 문
    int numberofdoors;   // 문 개수

    ~Map(); // 소멸자 선언
};
```

각 만들 사항에 대한 class를 정의 (class를 이용)

```
Map *CreateMap(int fleid, int numofdoor)
{
    Map *newmap = new Map();
    srand(time(NULL) + fleid);
    newmap->numberofdoors = numofdoor;

    switch (fleid)
    {
    case 0:
        newmap->coordinate = {0, 0};
        break;
    case 1:
        newmap->coordinate = {33, 0};
        break;
    case 2:
        newmap->coordinate = {66, 0};
        break;
    case 3:
        newmap->coordinate = {0, 20};
        break;
    case 4:
        newmap->coordinate = {33, 20};
        break;
    case 5:
        newmap->coordinate = {66, 20};
        break;
    }

    newmap->height = rand() % 6 + 4; // 최소 4 ~ 최대 9
    newmap->width = rand() % 14 + 4; // 최소 4 ~ 최대 17

    newmap->coordinate.x += rand() % (29 - newmap->width + 1);
    newmap->coordinate.y += rand() % (17 - newmap->height + 1);
```

```
    newmap->doors = new Door[numofdoor];
    for (int i = 0; i < numofdoor; i++)
    {
        newmap->doors[i].connect = 0;
        newmap->doors[i].coordinate = {0, 0};
    }

    // 위쪽문
    newmap->doors[0].coordinate.x = (rand() % (newmap->width - 2)) + 1 + (newmap->coordinate.x);
    newmap->doors[0].coordinate.y = newmap->coordinate.y;

    // 좌측문
    newmap->doors[1].coordinate.x = newmap->coordinate.x;
    newmap->doors[1].coordinate.y = (rand() % (newmap->height - 2)) + 1 + (newmap->coordinate.y);

    // 밑 문
    newmap->doors[2].coordinate.x = (rand() % (newmap->width - 2)) + 1 + (newmap->coordinate.x);
    newmap->doors[2].coordinate.y = (newmap->coordinate.y) + (newmap->height) - 1;
    // 우측문
    newmap->doors[3].coordinate.x = (newmap->coordinate.x) + (newmap->width) - 1;
    newmap->doors[3].coordinate.y = (rand() % (newmap->height - 2)) + 1 + (newmap->coordinate.y);

    return newmap;
}
```

Map *craeteMap (switch구문, for 반복문 이용)

입력 – int fleid : 몇 번째 장소인지 입력, int numofdoor : 그 장소의 문 개수

반환 값 – 만든 장소의 정보

결과 – 몇 번째 장소인지에 따라 장소의 시작점을 정하고 난수를 생성해 장소좌표 및 가로,세로길이를 만들어 장소의 정보를 저장함

```
void DrawMap(Map *map)
{
    int x, y;

    // 상단 하단 경계 그리기
    for (x = map->coordinate.x; x < map->coordinate.x + map->width; ++x)
    {
        mvprintw(map->coordinate.y, x, "-");                    // 상단
        mvprintw(map->coordinate.y + map->height - 1, x, "-"); // 하단
    }

    // 좌우 경계와 내부 그리기
    for (y = map->coordinate.y + 1; y < map->coordinate.y + map->height - 1; ++y)
    {
        mvprintw(y, map->coordinate.x, "|");                    // 좌측
        mvprintw(y, map->coordinate.x + map->width - 1, "|"); // 우측

        for (x = map->coordinate.x + 1; x < map->coordinate.x + map->width - 1; ++x)
        {
            mvprintw(y, x, ".");
        }
    }
    // 문 그리기
    for (int i = 0; i < 4; ++i)
    {
        mvprintw(map->doors[i].coordinate.y, map->doors[i].coordinate.x, "+");
    }
}
```

Void DrawMap (for 반복문 이용)

입력 – Map *map : 맵 클래스

반환 값 – 없음 (장소를 화면에 출력 – ncurses.h의 mvprintw를 이용)

결과 – map class안의 정보를 바탕으로 콘솔 화면에 장소를 출력

```
Map **MapSetUp()
{
    int setup = 6; // 장소 수 설정
    Map **maps = new Map *[setup];
    for (int i = 0; i < setup; i++)
    {
        maps[i] = CreateMap(i, 4);
        DrawMap(maps[i]);
    }
    return maps;
}
```

Map **mapSetup (for반복문 이용)

입력 – 없음

반환 값 – 새로운 Map객체 생성

결과 – 장소를 만들기 위한 프리셋을 만듦, ~~추후에 랜덤 장소 만들기 방식 채용 예정~~

Map class를 만들어 각 Map class에 정보를 만듦

```cpp
void ConnectDoor(Level *level) {
    srand(time(NULL));

    for (int i = 0; i < level->numberofmaps; i++) {
        for (int j = 0; j < level->map[i]->numberofdoors; j++) {
            if (level->map[i]->doors[j].connect == 1) {
                continue;
            }

            int count = 0;
            int maxAttempts = 10; // 문 연결 최대 시도 횟수
            bool connected = false;

            while (count < maxAttempts) {
                int randommap = rand() % level->numberofmaps;

                if (!level->map[randommap]) {
                    cerr << "오류: 장소 " << randommap << " 이 초기화 되어있지 않습니다." << endl;
                    count++;
                    continue;
                }

                int randomdoor = rand() % level->map[randommap]->numberofdoors;

                cout << "문 확인 : map[" << randommap << "] door[" << randomdoor << "] "
                     << "& map[" << i << "] door[" << j << "]" << endl;

                // 동일 맵이거나 이미 연결된 문이면 스킵
                if (level->map[randommap]->doors[randomdoor].connect == 1 || randommap == i) {
                    cout << "같은 맵에 있는 문이거나 연결된 문입니다." << endl;
                    count++;
                    continue;
                }
```

```cpp
                // 문 좌표 유효성 검사 및 재생성
                if (level->map[randommap]->doors[randomdoor].coordinate.x <= 0 ||
                    level->map[randommap]->doors[randomdoor].coordinate.x >= MAX_WIDTH ||
                    level->map[randommap]->doors[randomdoor].coordinate.y <= 0 ||
                    level->map[randommap]->doors[randomdoor].coordinate.y >= MAX_HEIGHT ||
                    level->map[i]->doors[j].coordinate.x <= 0 ||
                    level->map[i]->doors[j].coordinate.x >= MAX_WIDTH ||
                    level->map[i]->doors[j].coordinate.y <= 0 ||
                    level->map[i]->doors[j].coordinate.y >= MAX_HEIGHT) {
                    cerr << "오류 : 문 좌표가 유효하지 않은, 다시 시작중..." << endl;
                    count++;
                    continue;
                }

                // 경로 탐색
                try {
                    PathFind(&level->map[randommap]->doors[randomdoor].coordinate,
                             &level->map[i]->doors[j].coordinate);

                    // 연결 성공
                    level->map[randommap]->doors[randomdoor].connect = 1;
                    level->map[i]->doors[j].connect = 1;
                    connected = true;
                    break;
                } catch (const exception &e) {
                    cerr << "경로 탐색 중 오류 발생 :" << e.what() << endl;
                    count++;
                    continue;
                }
```

```cpp
                // 연결 실패 시 경고 메시지
                if (!connected) {
                    cerr << "map [" << j << "] 의 문 [" << i << "] 을 "
                         << maxAttempts << " 번 시도, 연결 실패." << endl;
                }
            }
        }
    }
```

void ConnectDoor (try-catch 구문 이용)

입력 – Level *level – 장소, 유저, 몬스터 등 각종 객체 정보가 있는 class

반환 값 – 없음(문들이 연결됨)

결과 – 각 장소의 문을 다른 랜덤한 문과 연결, 이때 같은 장소의 문, 이미 연결된 문일 경우 디버그 메시지 출력

(2) 플레이어 및 몬스터 구현

```
class Player
{
public:
    Position coordinate;    // 유저의 x,y좌표
    int health, maxhealth; // 유저의 체력
    string item;           // 유저의 장비
    int attack;            // 유저의 공격력
};

class Monster
{
public:
    char script;
    Position coordinate; // 몬스터의 x,y좌표
    int health;
    int attack;
    int speed;
    int pathfinding;
    int alive;
};
```

각 만들 사항에 대한 class를 정의 (class를 이용)

```
// 유저 시작 위치
void PlayerSetStartPosition(Map **map, Player *user)
{
    user->coordinate.x = map[3]->coordinate.x + 1;
    user->coordinate.y = map[3]->coordinate.y + 1;
}
```

void PlayerSetStartPosition

입력 – Map **map : 구현된 장소객체, Player *user : 만들 유저 객체

반환 값 – 없음

결과 – 장소 3번에 대해 유저 시작 좌표를 객체에 저장

```
Position *HandleInput(int input, Player *user)
{
    Position *newposition = new Position();

    switch (input)
    {
    // 위 이동
    case 'w':
    case 'W':
        newposition->x = user->coordinate.x;
        newposition->y = user->coordinate.y - 1;
        break;
    // 아래 이동
    case 's':
    case 'S':
        newposition->x = user->coordinate.x;
        newposition->y = user->coordinate.y + 1;
        break;
    // 왼쪽 이동
    case 'a':
    case 'A':
        newposition->x = user->coordinate.x - 1;
        newposition->y = user->coordinate.y;
        break;
    // 오른쪽 이동
    case 'd':
    case 'D':
        newposition->x = user->coordinate.x + 1;
        newposition->y = user->coordinate.y;
        break;
    default:
        delete newposition;
        break;
    }

    return newposition;
```

Position *HandleInput (switch구문 이용)

입력 – int input : 입력 값, Player *user : 유저 객체

반환 값 – 유저 객체의 새로운 좌표

결과 – 입력값에 따라 객체에 새로운 좌표를 줌

```c
void CheckPosition(Position *position, Level *level)
{
    Player *user = level->player;
    int space;
    switch (mvinch(position->y, position->x))
    {
    // 갈 수 있는 장소면 이동
    case '.':
    case '#':
    case '+':
        PlayerMove(position, user);
        break;
    // 적이면 전투 발생
    case 'S':
    case 'G':
    case 'H':
        Combat(user, GetTileMonster(position, level->monster), 1);
    default:
        break;
    }
}
```

Void checkPosition (switch 구문 이용)

입력 – Position *position – 새 좌표에 관한 class, Level *level – 장소, 유저, 몬스터 등 각종 객체 정보가 있는 class

반환 값 – 없음

결과 – 유저의 새로운 좌표가 무엇인지에 따른 행동 수행

```
void PathFindingRandom(Position *coordinate)
{
    srand(time(NULL));
    int randomnum;
    randomnum = rand() % 5;
    switch (randomnum)
    {
    case 0: // 상단 이동
        if (mvinch(coordinate->y - 1, coordinate->x) == '.')
        {
            coordinate->y = coordinate->y - 1;
        }
        break;

    case 1: // 하단 이동
        if (mvinch(coordinate->y + 1, coordinate->x) == '.')
        {
            coordinate->y = coordinate->y + 1;
        }
        break;
    case 2: // 좌측 이동
        if (mvinch(coordinate->y, coordinate->x - 1) == '.')
        {
            coordinate->x = coordinate->x - 1;
        }
        break;
    case 3: // 우측 이동
        if (mvinch(coordinate->y, coordinate->x + 1) == '.')
        {
            coordinate->x = coordinate->x + 1;
        }
        break;
    case 4: // 제자리
        break;
    }
}
```

void PathFindingRandom (switch 구문 이용)

입력 – Position * coordinate – 좌표에 관한 객체class

반환 값 – 없음 (몬스터 이동)

결과 – 난수를 생성하여 상황에 맞게 몬스터가 랜덤 이동

```
void PathFindingChase(Position *start, Position *end)
{
    if ((abs((start->x - 1) - end->x) < abs(start->x - end->x)) && (mvinch(start->y, start->x - 1) == '.'))
    {
        start->x = start->x - 1; // 좌측 이동
    }
    else if ((abs((start->x + 1) - end->x) < abs(start->x - end->x)) && (mvinch(start->y, start->x + 1) == '.'))
    {
        start->x = start->x + 1; // 하단 이동
    }
    else if ((abs((start->y + 1) - end->y) < abs(start->y - end->y)) && (mvinch(start->y + 1, start->x) == '.'))
    {
        start->y = start->y + 1; // 하단 이동
    }
    else if ((abs((start->y - 1) - end->y) < abs(start->y - end->y)) && (mvinch(start->y - 1, start->x) == '.'))
    {
        start->y = start->y - 1; // 상단 이동
    }
}
```

void PathFindingChase

입력 – Position *start – 몬스터가 이동하기전 좌표, Position *end – 몬스터가 이동하게 될 (유저의 좌표)좌표

반환 값 – 없음

결과 – 목적지 좌표와 현재 좌표가 이동하게 될 거리를 계산하고 그 자리가 이동 가능 좌표면 이동

(3) 유저 정보 출력

```
void CoutGameInformation(Level *level)
{
    if (!level || !level->player)
    {
        cerr << "오류 : 레벨이나 유저가 초기화되지 않음" << endl;
        return;
    }

    cerr << "디버그 : Level = " << level->level
         << ", Player Hp = " << level->player->health
         << ", Max Hp = " << level->player->maxhealth
         << ", Attack = " << level->player->attack << endl;

    refresh();
    mvprintw(45, 1, "Level: %d | Hp: %d(%d) | Attack: %d      ",
             level->level, level->player->health,
             level->player->maxhealth, level->player->attack);
    cout << "디버그 : 정보가 불러와짐" << endl;
    refresh();
}
```

void CoutGameInformation (cerr로 디버그 메시지 출력)

입력 – Level *level – 장소, 유저, 몬스터 등 각종 객체 정보가 있는 class

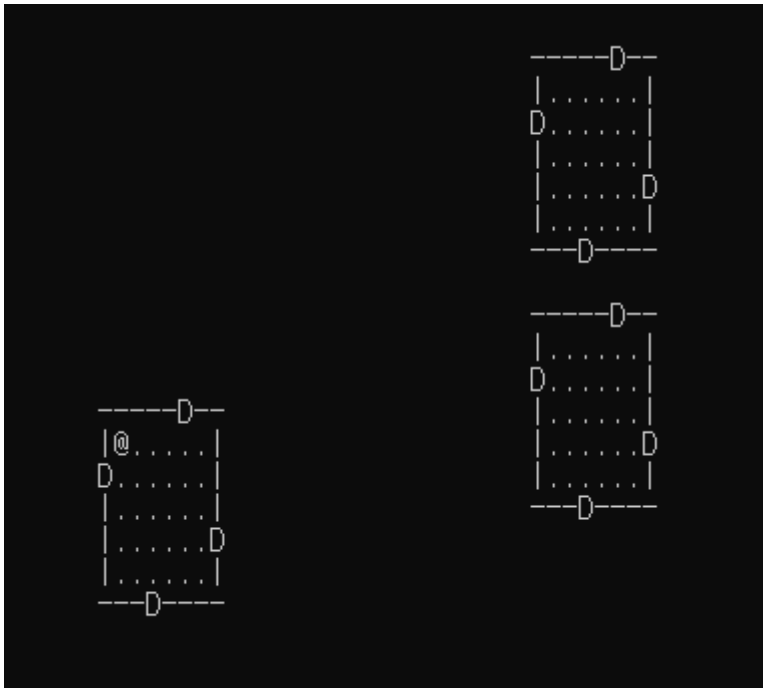
반환 값 – 없음

결과 – 유저 정보가 출력됨

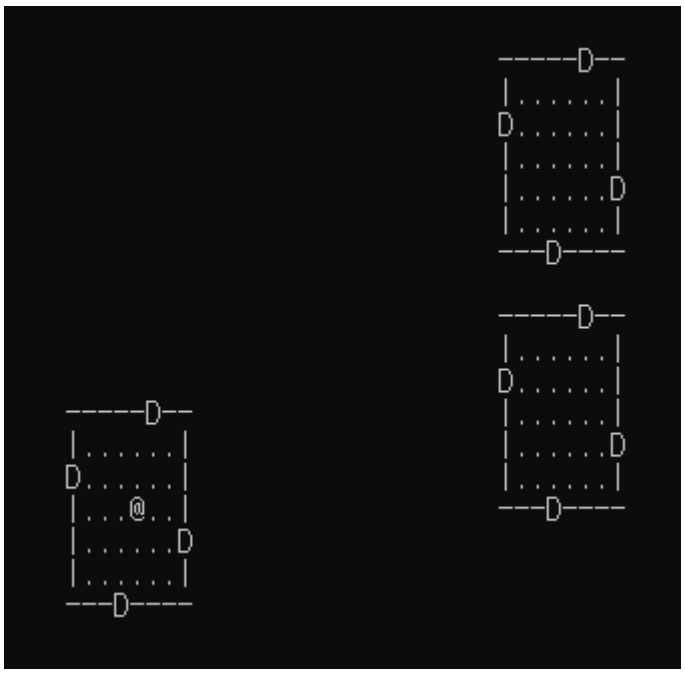
2) 테스트 결과

(1) 장소 생성과 문 생성

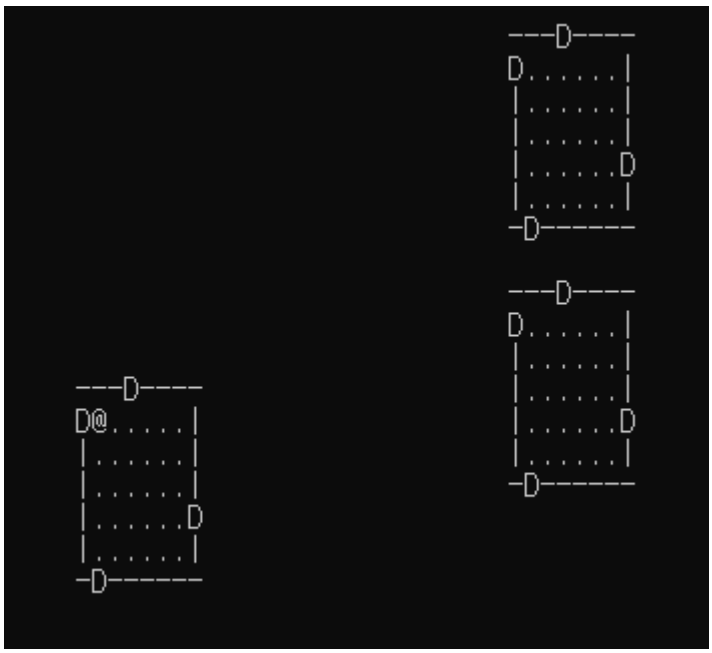
- 실행파일 실행 시 장소를 생성(. , | , _ 로 장소를 구성, D로 문 구성, @로 유저 구성)



- wasd로 @인 유저 이동

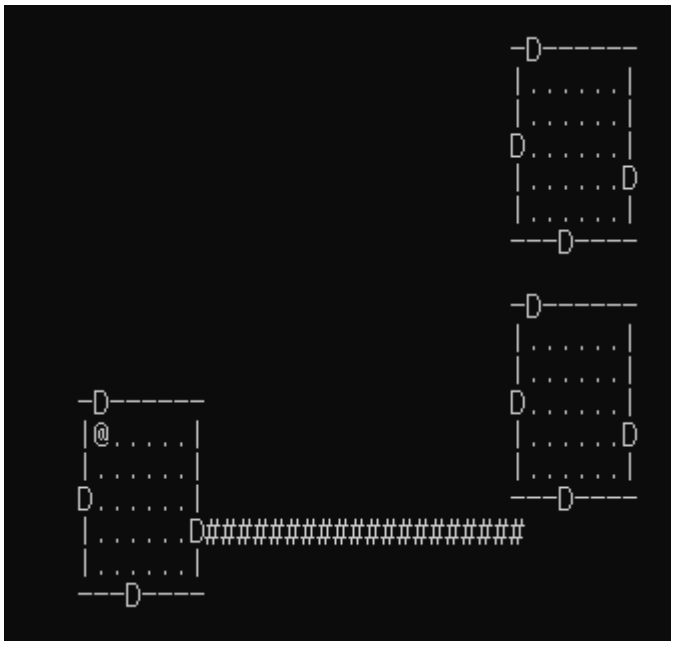


- 실행파일 실행 시 마다 장소에 문이 랜덤 생성



(2) 장소 연결

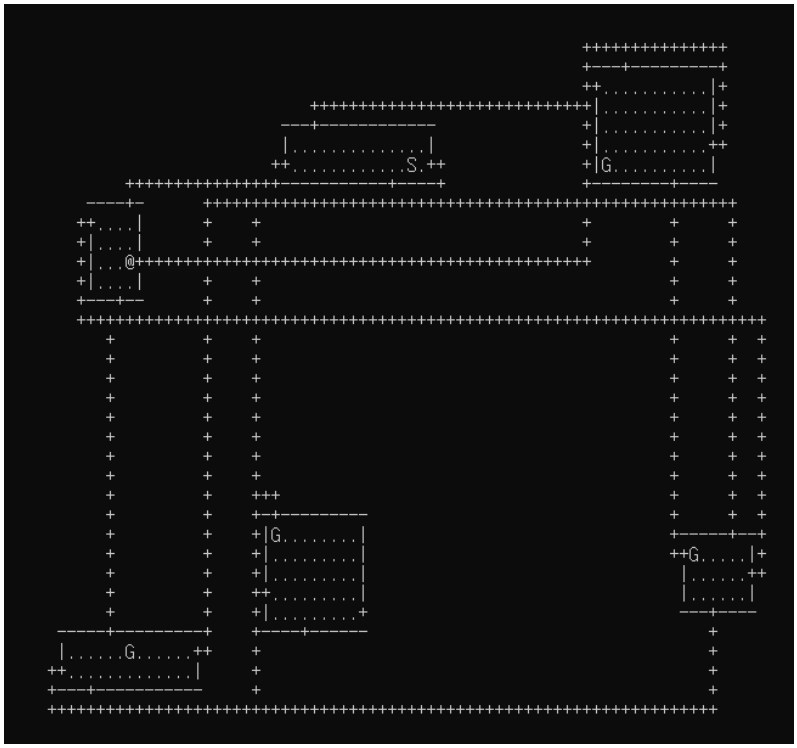
- 장소에 문 연결



(3) 매 시도 변화

- 장소 및 몬스터 랜덤 생성

- 몬스터의 랜덤 이동 및 추적 이동(S는 랜덤 이동, G는 추적 이동)



- 유저 정보 및 갱신

```
Level: 1 | Hp: 20(20) | Attack: 1
```

```
Level: 1 | Hp: 19(20) | Attack: 1
```

4. 계획 대비 변경 사항

1) 1차 일정 변경

이전 – 유저의 이동 방식 구현을 map을 만든 다음 하려고 시도

이후 – map생성과 유저의 이동방식 구현을 동시에 진행하고 무작위 map생성 일정을 뒤로 미룸

사유 – map생성을 해보니 원하는 방향으로 설계된 것인지 확인을 위해 테스트용 유저를 만들 필요가 있었고 무작위로 map생성 구현이 생각보다 시간이 걸려 순서를 바꾸기로 함

2) 2차 일정 변경

이전 – 여러 헤더 파일을 만들어 각각에 대응하는 코드를 구현

이후 – 헤더 파일 통합 후 나머지 사항 구현

사유 – 헤더파일을 여러 개로 하니 어디가 어디 있는지 가독성이 떨어져 구현하는데 어려움 발생 또한 장소에 속성을 부여하는 코드와 화면에 실제로 프린트 되는 계산에 문제가 생겨 고치는 중이다.

3) 3차 일정 변경

변동사항 없음

5. 프로젝트 일정

업무		11/3	11/10	11/17	11/24	12/1	12/8	12/15	12/22
제안서 작성		완료							
기능1	맵 생성	완료							
	맵 구현		완료						
기능2	Player 이동			완료					
	Enemy 이동			완료					
기능3					진행 중 ---------->				
계산 논리 고치기					완료				
기능4	시작화면					진행 중 ----->			
	시작장소					진행 중 ----->			
기능5	일시정지						진행 중 ----->		
	상태화면							완료	
최종 점검 및 제출									진행 중