

C/C++ 텍스트 게임 만들기 (보물찾기)

2018875036 송준엽

개요

1. 기존 구현된 게임 분석
2. 추가 구현할 방향 제시
3. 구현한 UI설명(게임의 큰 틀)
4. 구현한 기능 설명(게임의 부분요소, 알고리즘)
5. 게임 외 코드관리
6. 참고자료

기존 구현된 게임 분석(1)

- Void display_rule();
 - - 게임 룰 설명해주는 프로시저
- Void basic_map();
 - - ■ 문자를 10x10크기로 출력해주는 프로시저
- Void display_map(int matrix[][11], int tx[], int ty[])
 - -basic_map()을 호출하고 보물의 위치, matrix의 방문여부에 따라 출력을 달리함

기존 구현된 게임 분석(2)

```
void make_treasure(int tx[], int ty[])
{
    int i;
    for (i = 0; i < 2; i++)
    {
        do
        {
            tx[i] = rand() % 11; //보물의 x좌표
            ty[i] = rand() % 11; //보물의 y좌표
        } while ((tx[i] <= 1) || (ty[i] <= 1));
    }
}
```

0~10까지 랜덤한 x,y 좌표 값을 설정해서 tx[], ty[]에 2번 넣어주는 함수 입니다.

※이때 첫번째 보물의 좌표와 두번째 보물의 좌표의 중복검사는 하지 않습니다.(수정)

추가 구현할 방향 제시(1)

- 위에 게임을 분석한 결과는
- 무작위 위치에 숨겨져 있는 보물(2개)을 유저의 입력으로
- 찾아 나가고, 찾는데 걸린 최종 시간 출력해주는 게임입니다.

1. 위 게임을 플레이 해 보면서 아쉬운 점은 '긴장감을 부여하는 게임 요소가 더 많았으면 어땠을까?'라는 생각과

2. '한 스테이지를 하고나서 다음 스테이지를 하고싶게 하는 원동력을 추가하면 어떨까?' 라는 생각을 했습니다.

추가 구현할 방향 제시(2)

위 1번을 토대로 추가로 구현해 볼 기능

- 맵에 벽 추가, 몬스터 만들기, 아이템 만들기, 시간제한

위 2번을 토대로 추가로 구현해 볼 기능

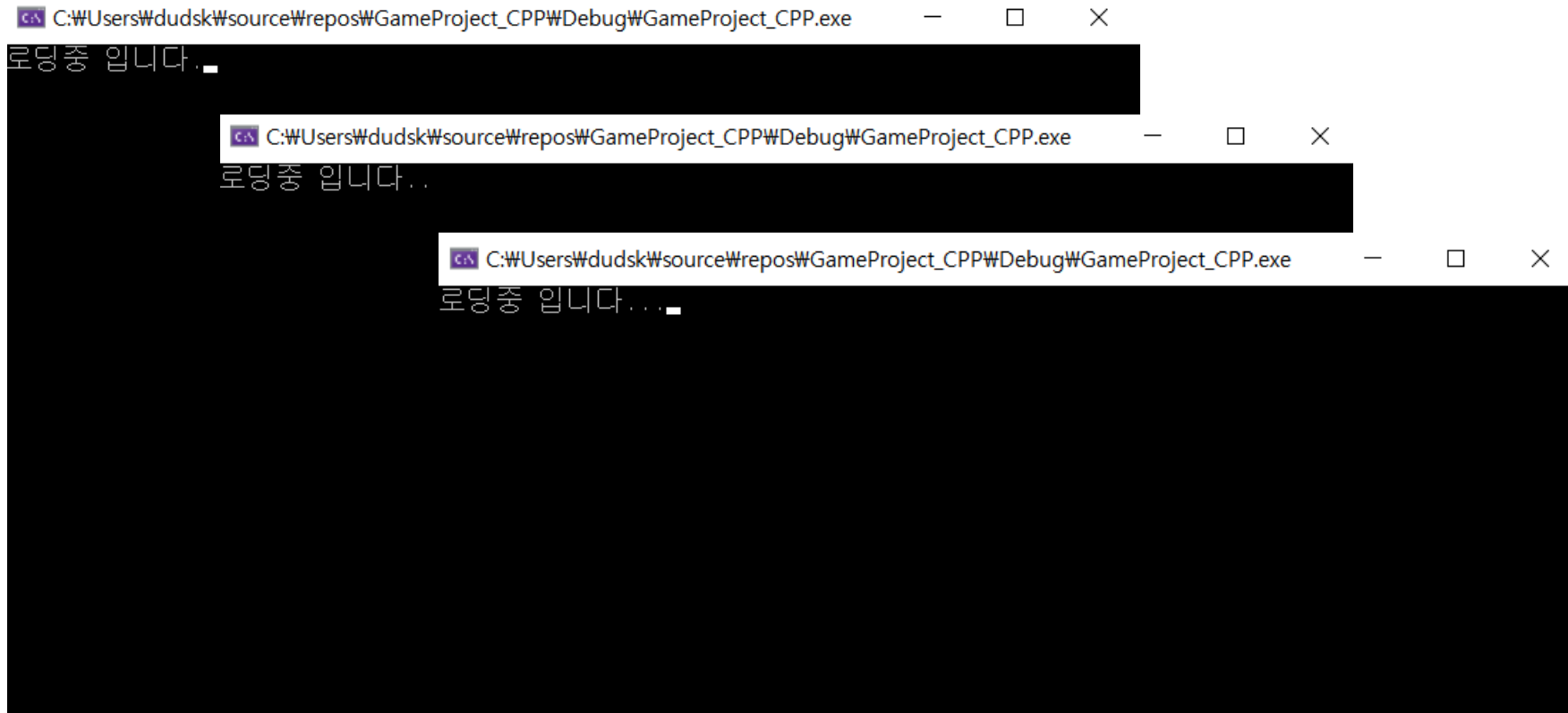
- 상점 구현(메인메뉴), 상점을 이용할 수 있는 재화 추가, 스테이지 만들기

기존 게임 : 랜덤으로 숨겨진 보물을 찾아나감

새 게임 : 몬스터를 피해(잡고) 숨겨진 열쇠를 찾고 보물상자를 열고 나감

구현할 UI 설명(1)

- UI설명에 앞서 콘솔 창의 크기를 30 x 80으로 설정했습니다
- `Int main() { system("mode con: cols=80 lines=30"); ...}`
- (1)로딩 화면



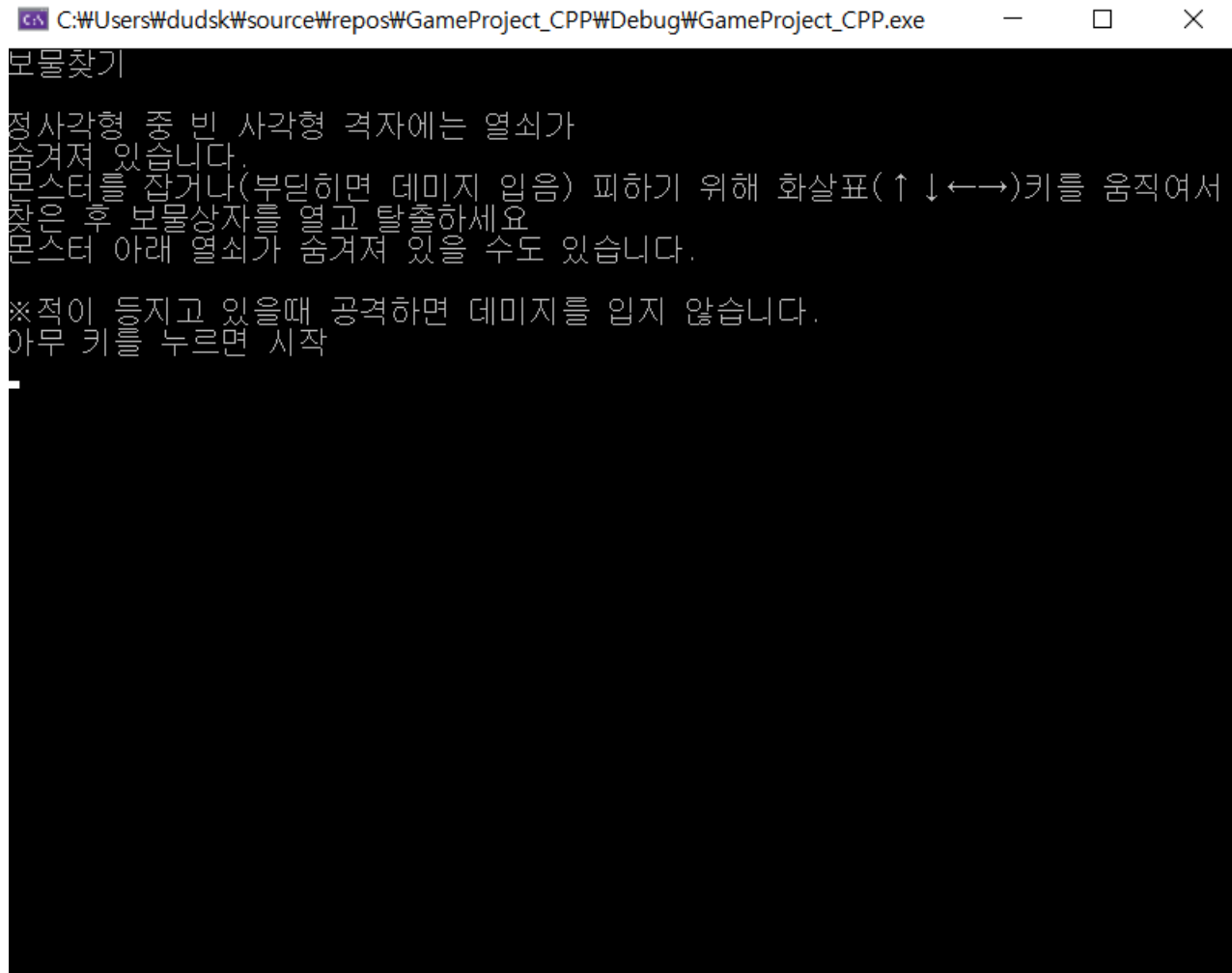
구현할 UI 설명(2)

- (2)메인 화면



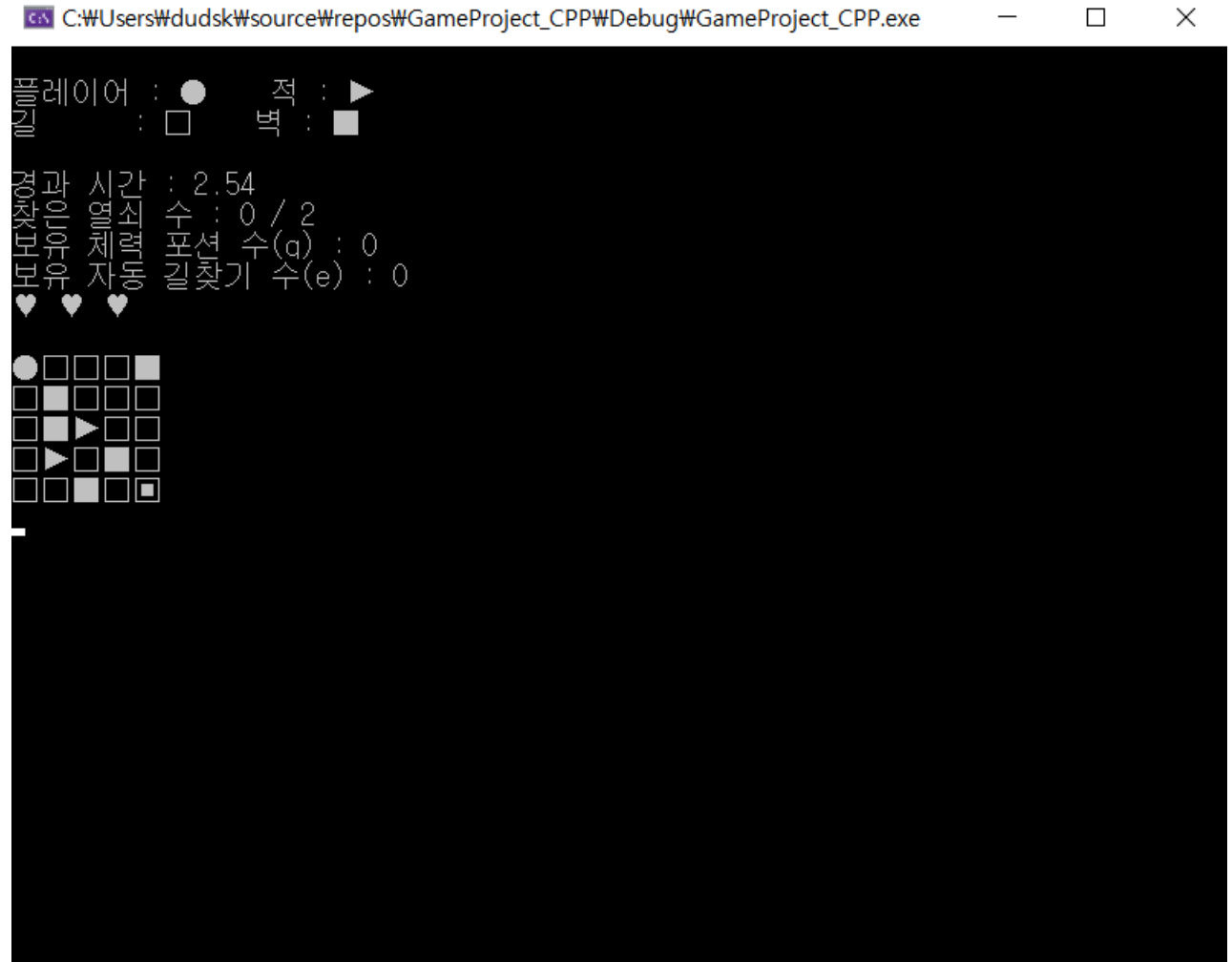
구현할 UI 설명(3)

- (3)게임 설명 화면



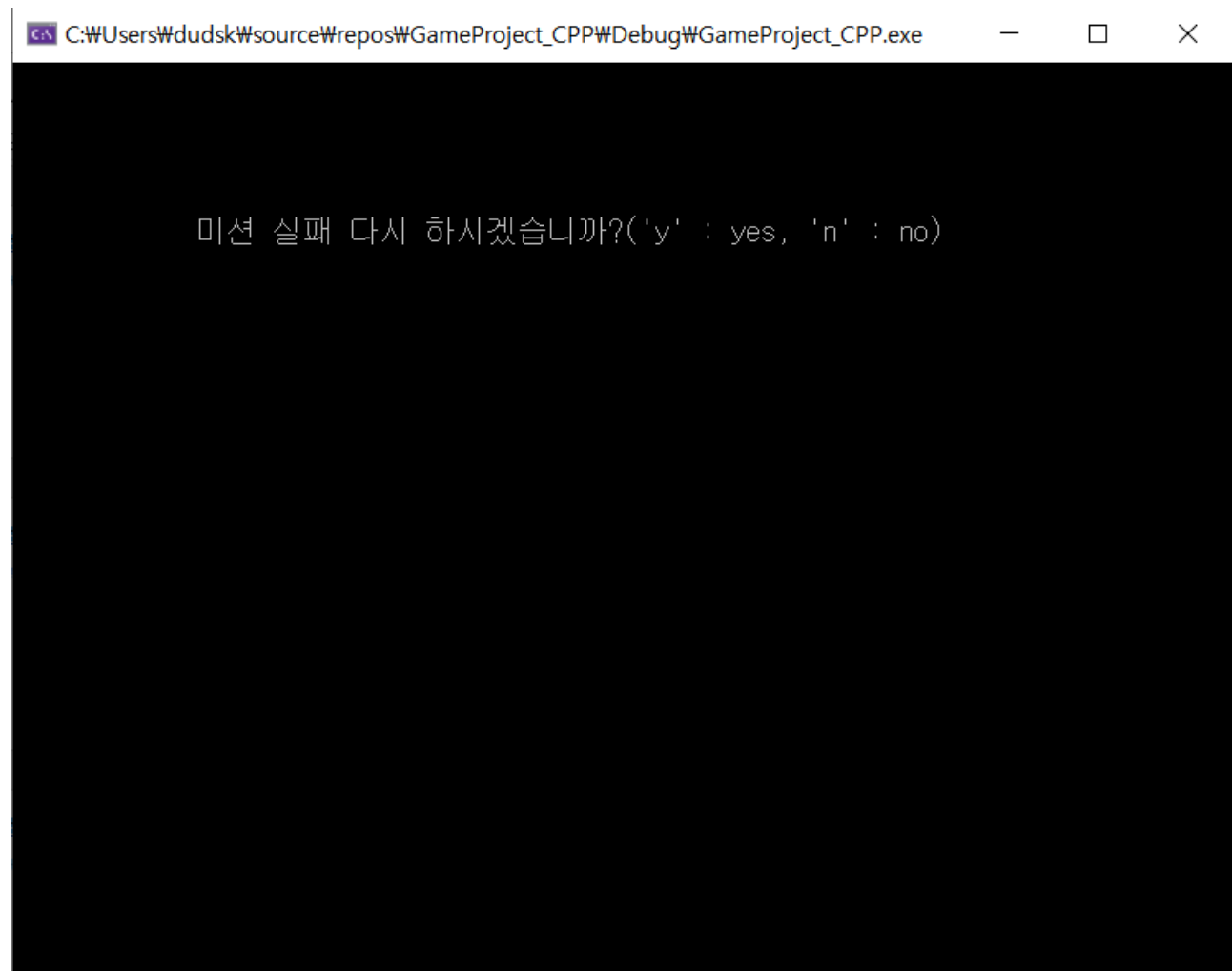
구현할 UI 설명(4)

- (4)게임 화면



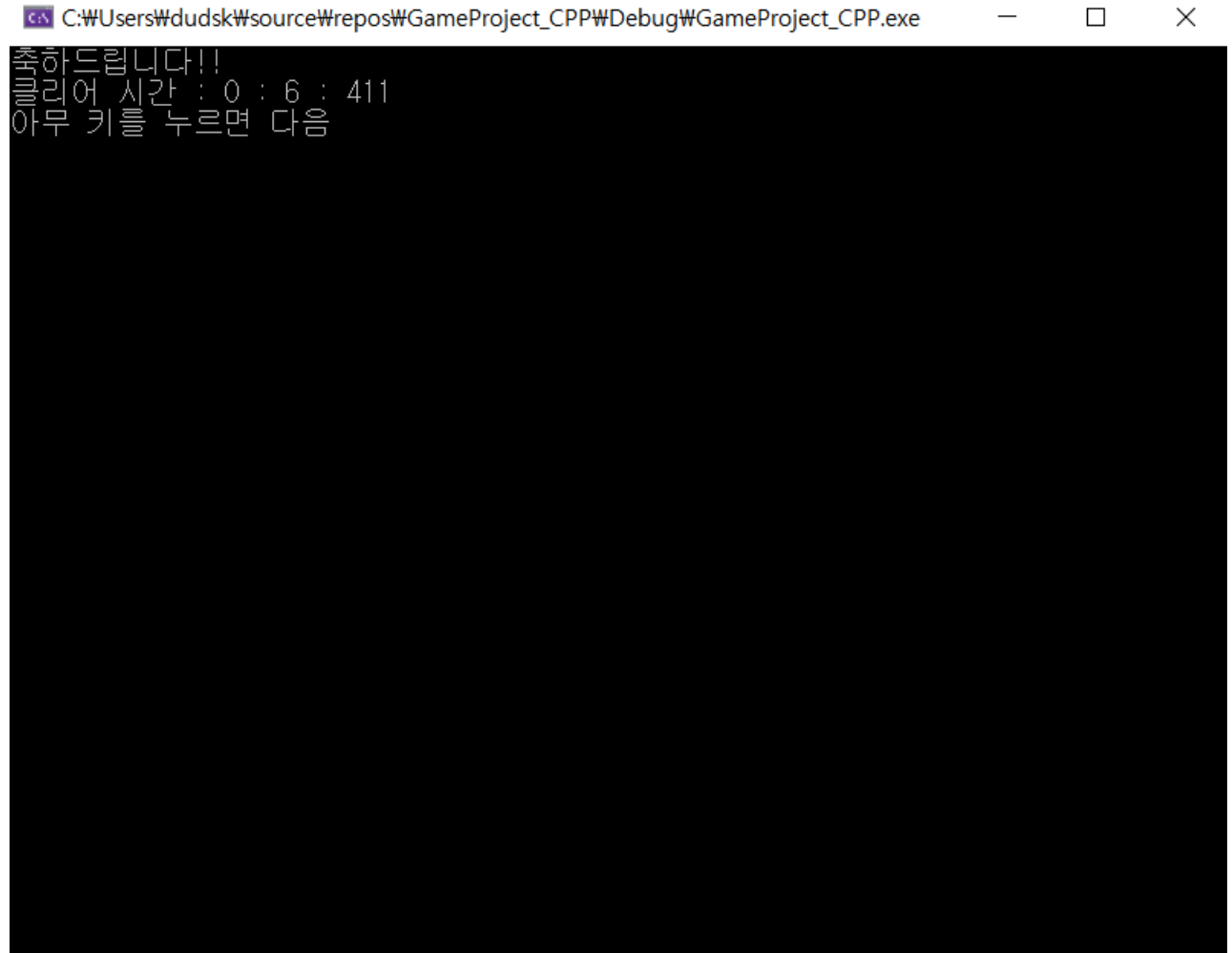
구현할 UI 설명(5)

- (4.1)게임 실패 화면



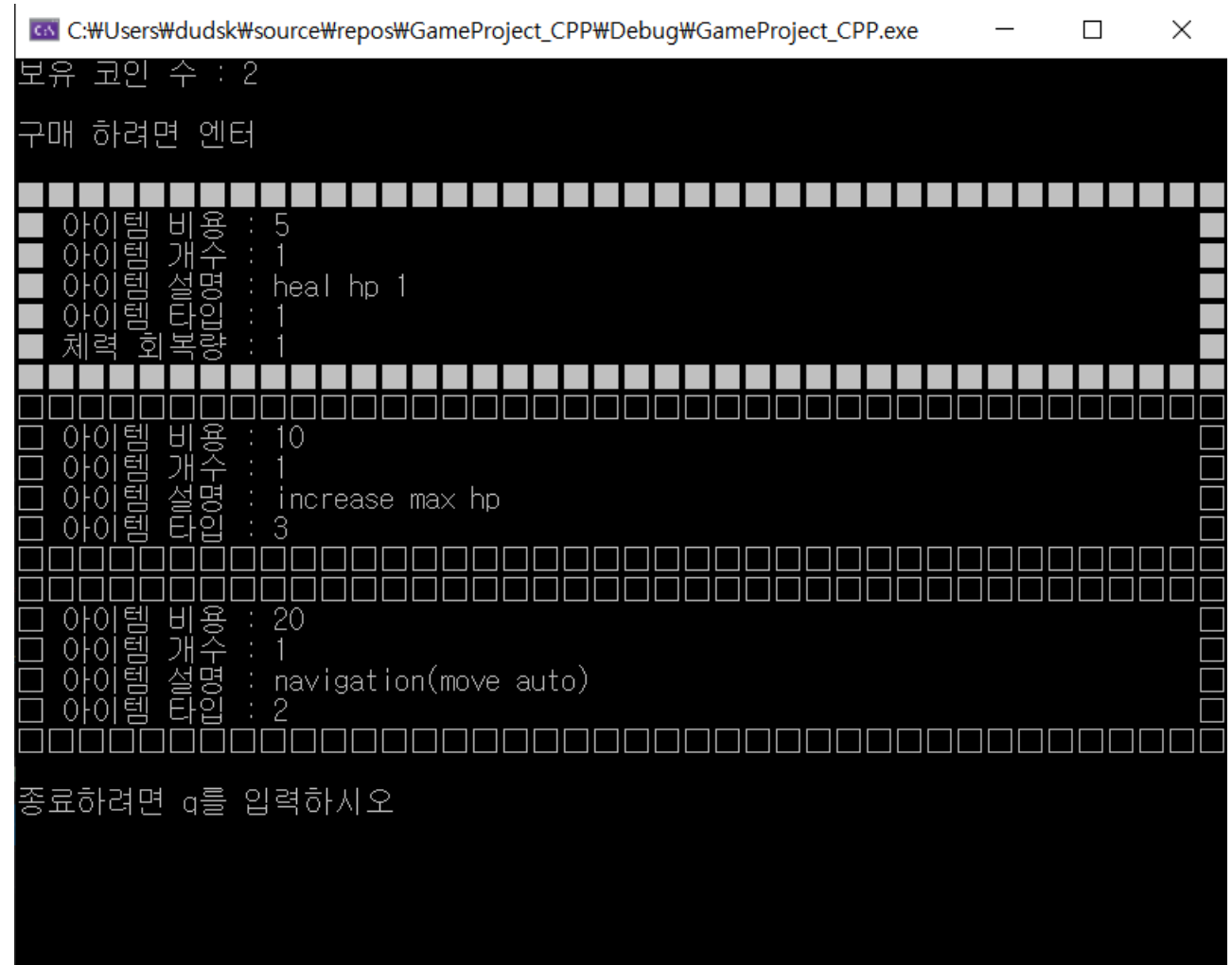
구현할 UI설명(6)

- (4.2)게임 성공 화면



구현할 UI설명(7)

- (5)상점 화면



The screenshot shows a terminal window titled "C:\Users\dudsk\source\repos\GameProject_CPP\Debug\GameProject_CPP.exe". The interface is a text-based shop menu. At the top, it says "보유 코인 수 : 2" (Current coin count: 2) and "구매 하려면 엔터" (Press Enter to purchase). Below this is a list of items, each preceded by a checkbox and followed by its details: name, cost, quantity, and description. The items are:

- ☐ 아이템 비용 : 5, 개수 : 1, 설명 : heal hp 1, 타입 : 1, 체력 회복량 : 1
- ☐ 아이템 비용 : 10, 개수 : 1, 설명 : increase max hp, 타입 : 3
- ☐ 아이템 비용 : 20, 개수 : 1, 설명 : navigation(move auto), 타입 : 2

At the bottom, it says "종료하려면 q를 입력하시오" (Press q to exit).

구현한 기능 설명(0)

- 각종 매크로, 구조체 및 열거형

```
#define BILLION 1000000000
#define STAGE 1
#define ALLITEMSIZE 3
#define INVENSIZE 3
enum MENU_LIST
{
M_NONE,
PLAY,
STORE,
EXIT,
};
enum ITEM_TYPE {
I_NONE,
POTION,
CONSUME,
ETC,
};
```

```
struct Item {
char* item_code;
char* description;
int cost;
int count;
int hp_recovery;

ITEM_TYPE item_type;
};
```

```
struct BaseInfo {
int menu_list_size;
MENU_LIST menu_list;
};
```

```
struct BaseCharacter {
int health;
int max_health;
int coin;

int pos[2];
};
```

```
struct Monster : public BaseCharacter
{
int turn_cycle;
int max_turn_cycle;
};
```

```
struct PlayerCharacter : public BaseCharacter
{
int taked_key_size;
```

```
Item** inventory;
};
```

```
struct MapInfo {
int map_size;
int wall_size;
int enemy_size;
int key_size;
int key_pos[20][3]; // 3번째 인덱스에는 해당 키가 찾아졌는지 확인
float time_limit;
int play_time;
```

```
Monster** monsters;
Land** lands;
};
```

구현한 기능 설명

(1.0 자동으로 맵 만들기)

```
bool initialize_BMC(BaseInfo* base_info, MapInfo** map_infos, PlayerCharacter* player_info)
{
    clock_t start = clock();
    Float min_loading_time = 1000;

    Base_info->menu_list = MENU_LIST::PLAY;
    Base_info->menu_list_size = 3;

    For (int i = 0; i < STAGE; i++) {
        Map_infos[i] = (MapInfo*)malloc(sizeof(MapInfo));
        Int map_size = 5 * (i / 3 + 1);
        Int wall_size = map_size * map_size / 5;
        Int key_size = map_size / 5 + i % 3 + 1;
        Make_map(map_infos[i], map_size, key_size, wall_size, key_size, map_size * 3);
    }

    initialize_player_character(player_info, 3, 3);
    float running_time = clock() - start;
    if (running_time < min_loading_time) {
        Sleep(min_loading_time - running_time);
    }
    return true;
}
```

게임실행 시 초기화 되어야하는 정보(화면 구성 요소, 맵 정보, 유저 정보)를 초기화 하는함수

이 함수는 make_map(...)에 따라 실행시간이 길어질 수도 있어 비동기로 실행이 되는데 이때 (1)로딩화면이 출력됩니다.-

만약 옆 함수의 실행시간이 min_loading_time보다 작다면 min_loading_time시간이 될 때 까지 Sleep(...)를 실행합니다.

구현한 기능 설명

(1.1 자동으로 맵 만들기)

```
void make_map(MapInfo* map_info, int size, int enemy, int wall, int key,
float time_limit)
{
    ... // 기본 초기화 코드

    map_info->lands = (Land**)malloc(sizeof(Land*) * size);
    for (int i = 0; i < size; i++) {
        map_info->lands[i] = (Land*)malloc(sizeof(Land) * size);
    }
    ... // 기본 초기화 코드
    make_wall(map_info, wall);
    while (!AStar(map_info, 0, 0, size - 1, size - 1)) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                set_land_type(&map_info->lands[i][j], LAND_TYPE::LOAD);
            }
        }
        make_wall(map_info, wall);
    }
}
```

0,0 좌표(플레이어 위치)와 n, n 좌표(보물상자 위치)가 벽에 의해 이어지지 않으면 벽을 재 생성 합니다.

```
make_key(map_info, key);
for (int i = 0; i < key; i++) {

    if (!AStar(map_info, map_info->key_pos[i][0], map_info->key_pos[i][1], size -
1, size - 1)) {
        make_key(i, map_info);
        i--;
    }
}

map_info->monsters = (Monster**)malloc(sizeof(Monster*) * enemy);
for (int i = 0; i < map_info->enemy_size; i++) {
    map_info->monsters[i] = (Monster*)malloc(sizeof(Monster));
    int max_health = 1;
    int max_cycle = random(2, 5);
    initialize_Monster(map_info->monsters[i], max_health, max_health, max_cycle);
}

set_spawn_pos(map_info);
}
```


구현한 기능 설명

(1.2 자동으로 맵 만들기)

```
enum LAND_TYPE
{
    LOAD,
    WALL,
    START,
    END,
};
```

```
struct Land {
    int pos_x, pos_y;
    int F, G, H; //f는 기대비용 + 사용비용, g는 사용비용, h는 기대비용
    LAND_TYPE land_type;
    int visit_info; //0이면 방문x, 1이면 열기, 2이면 닫기
    Land* pre_land;
};
```

```
struct cmp {
    bool operator()(const Land* l1, const Land* l2) {
        return l1->F > l2->F;
    }
};
```

A*알고리즘에 쓰인 구조체

구현한 기능 설명

(1.3 자동으로 맵 만들기)

```
bool AStar(MapInfo* map_info, int pos_x, int pos_y, int dest_x, int dest_y) {  
    if (map_info == nullptr) return false;
```

```
    std::priority_queue<Land*, std::vector<Land*>, cmp> pq;  
    pq.push(&map_info->lands[pos_y][pos_x]);  
    map_info->lands[pos_y][pos_x].F = 0;  
    map_info->lands[pos_y][pos_x].G = 0;  
    map_info->lands[pos_y][pos_x].H = 0;
```

```
    //상 하 좌 우  
    int dx[] = { 0, 0, -1, 1 };  
    int dy[] = { -1, 1, 0, 0 };  
    int max_idx = map_info->map_size - 1;
```

```
    while (!pq.empty()) {  
        Land* cur_land = pq.top();  
        pq.pop();
```

```
        if (cur_land->visit_info == 2) continue;  
        //방문한 노드 닫기  
        cur_land->visit_info = 2;  
        //목표 노드이면 스택  
        if (cur_land->pos_x == dest_x && cur_land->pos_y == dest_y) {  
            reset_FGH(map_info);  
            return true;  
        }  
    }
```

```
    //상, 하, 좌, 우 체크  
    for (int i = 0; i < 4; i++) {  
        int tmp_x = cur_land->pos_x + dx[i];  
        int tmp_y = cur_land->pos_y + dy[i];  
        //배열범위 체크  
        if (tmp_x < 0 || tmp_y < 0 || tmp_x > max_idx || tmp_y > max_idx) continue;  
        if (map_info->lands[tmp_y][tmp_x].visit_info == 2) continue;  
        if (map_info->lands[tmp_y][tmp_x].land_type == LAND_TYPE::WALL) continue;
```

```
    //기존 f 보다 새 f가 더 작으면 교체  
    int new_g = cur_land->G + 10;  
    int new_h = (abs(dest_x - tmp_x) + abs(dest_y - tmp_y)) * 5;  
    if (new_g + new_h < map_info->lands[tmp_y][tmp_x].F) {  
        map_info->lands[tmp_y][tmp_x].F = new_g + new_h;  
        map_info->lands[tmp_y][tmp_x].G = new_g;  
        map_info->lands[tmp_y][tmp_x].H = new_h;  
        map_info->lands[tmp_y][tmp_x].pre_land = cur_land;  
    }
```

```
    //체크한 자리 열기(방문처리)  
    map_info->lands[tmp_y][tmp_x].visit_info = 1;  
    //체크한 자리 큐에 넣기  
    pq.push(&map_info->lands[tmp_y][tmp_x]);  
}
```

```
    reset_FGH(map_info);  
    return false;  
}
```

Land를 방문하고, 주변에 land의 f값을 갱신 후 우선순위 큐에 삽입합니다. 이후 방문하지 않고 갱신된 F의 값이 가장 작은 land를 방문합니다. 이때 land가 목표 노드이면 true를 리턴해 줍니다.

구현한 기능 설명

(2.0 플레이어의 이동)

```
Void move_arrow_key(char key, PlayerCharacter* user, MapInfo* map_info, float elapsed_time)
{
    Int tmp_x = user->pos[0], tmp_y = user->pos[1];

    ... //플레이어의 움직임과 아이템 사용

    If (tmp_x < 0 || tmp_x > map_info->map_size - 1) return;
    If (tmp_y < 0 || tmp_y > map_info->map_size - 1) return;
    If (map_info->lands[tmp_y][tmp_x].land_type == LAND_TYPE::WALL) return;

    Int m_idx = check_monster(map_info, tmp_x, tmp_y);
    //몬스터가 있는경우
    if (m_idx != -1 && map_info->monsters[m_idx]->health != 0) {
        take_damage(1, user, map_info->monsters[m_idx]);
    }
```

```
//플레이어의 체력감소
int idx = (map_info->monsters[m_idx]->turn_cycle / map_info->monsters[m_idx]-
>max_turn_cycle) % 4;
int dx[] = { -1, 0, 1, 0 };
int dy[] = { 0, 1, 0, -1 };
//몬스터의 공격권에 있지 않은지 판단
if (!(tmp_x + dx[idx] == user->pos[0] && tmp_y + dy[idx] == user->pos[1])) {
    take_damage(1, map_info->monsters[m_idx], user);
}
}

//움직일때 마다 사이클이 돌아 몬스터가 랜덤하게 돔
for (int i = 0; i < map_info->enemy_size; i++) {
    map_info->monsters[i]->turn_cycle++;
}

user->pos[0] = tmp_x, user->pos[1] = tmp_y;
}
```

플레이어의 키입력에 따라 이동하는데 이동이후 위치가 벽이거나 몬스터 이면 두 객체가 상호 작용 합니다. 만약 플레이어가 몬스터의 평평한 부분과 닿게 되면 플레이어는 피해를 입지 않습니다. 그리고 플레이어의 키입력이 일정 수가 되면 몬스터의 보는 방향이 달라집니다.

구현한 기능 설명

(2.1 플레이어의 이동)

```
int take_damage(int damage, BaseCharacter* anti_target, BaseCharacter* target)
{
    target->health = clamp(target->health - damage, 0, target->max_health);
    if ((Monster*)target != nullptr && target->health == 0) {
        anti_target->coin += target->coin;
    }
    return damage;
}
```

공격을 한 대상자와, 피대상자를 파라미터로 받아와 피대상자의 체력을 감소시키고 만약 공격을 입은 대상이 몬스터 이면 플레이어의 코인을 증가 시킵니다. 이때 targe을 (Monster*)구조체로 형변환을 해 몬스터 인지 판단해 줍니다.

구현한 기능 설명

(3.0 게임 로직)

```
bool game_control(MapInfo* map_info, PlayerCharacter* user, float start_time)
{
Do {
    Int user_pos_x = user->pos[0];
    Int user_pos_y = user->pos[1];
    Float elapsed_time = (clock() - start_time) / CLOCKS_PER_SEC;

    Check_key(map_info, user);

    //보물상자 위에 있는지 확인
    if (user_pos_x == map_info->map_size - 1 && user_pos_y == map_info->map_size - 1) {
        if (map_info->key_size == user->taked_key_size) {
            user->pos[0] = 0, user->pos[1] = 0;
            user->taked_key_size = 0;
            map_info->play_time = clock() - start_time;
            return true;
        }
    }
}
```

```
//미션 실패
if (fail_condition(map_info, user, elapsed_time)) {
    user->pos[0] = 0;
    user->pos[1] = 0;
    user->taked_key_size = 0;
    user->health = user->max_health;
    return false;
}

char input = user_input();
if (input != -1) {
    move_arrow_key(input, user, map_info, elapsed_time);
}
system("cls");
display_game(map_info, user, elapsed_time);
} while (true);
}
```

현재 맵과 유저 정보를 파라미터로 받아와 현재위치에 키가 있는지, 보물상자가 있는지 확인합니다.

그리고 해당 스테이지의 성공, 실패를 판단하고 데이터를 초기화 해 줍니다. 이후 키입력을 받아 다음 동작을 수행하고 게임화면을 전시해 줍니다.

구현한 기능 설명

(3.1 게임 로직)

```
bool fail_condition(const MapInfo* map_info, const BaseCharacter* user, float
elapsed_time)
{
    if (elapsed_time > map_info->time_limit ||
        user->health == 0) return true;

    return false;
}
```

이전의 함수(game_control)에서 스테이지의 경과 시간을 파라미터로 받아와서 제한시간을 초과하거나 유저의 체력이 0이 되면 false를 리턴해 줍니다

구현한 기능 설명

(3.2 게임 로직)

```
int user_input()
{
    if(kbhit())
        return getch();
    return -1;
}
```

game_control(...)함수를 구현할 때 경과시간을 계속 갱신 하면서 화면에 전시하고 싶었지만 getch()함수는 유저의 입력을 기다립니다. 따라서 유저가 입력을 하지 않으면 게임 화면이 바뀌지 않았는데 kbhit()은 현재 유저가 키를 누르고 있는지 판단해서 값을 리턴 해 줍니다. 따라서 kbhit() 함수를 사용하면 유저의 키입력과 상관없이 그 즉시 값을 리턴 해 주기 때문에 대기 시간이 없어집니다.

구현한 기능 설명

(4.0 아이템)

```
void make_item(Item** item)
{
    if (item == nullptr) return;

    for(int i = 0; i < ALLITEMSIZE; i++)
        item[i] = (Item*)malloc(sizeof(Item));
    char* c1 = "Item_HPPotion";
    char* d1 = "heal hp 1";
    initialize_item(item[0], c1, d1, 5, 1, 1, ITEM_TYPE::POTION);
    char *c2 = "Item_Increase_Max_HP";
    char *d2 = "increase max hp";
    initialize_item(item[1], c2, d2, 10, 1, 1, ITEM_TYPE::ETC);
    char *c3 = "Item_AStar";
    char *d3 = "navigation(move auto)";
    initialize_item(item[2], c3, d3, 20, 1, 0, ITEM_TYPE::CONSUME);
}
```

아이템을 만드는 함수입니다. 해당 함수에서 아이템을 정의 해야 합니다.

※const char*를 char*로 변환할 수 없습니다. 라는 오류 메시지가 출력될 텐데
프로젝트->속성->c/c++->언어->준수모드:false로 두면 됩니다.

구현한 기능 설명

(4.1 아이템)

```
bool buy_item(Item* item, PlayerCharacter* user)
{
    if (user->coin < item->cost) {
        return false;
    }

    int idx = 0;
    if ((idx = find_item_idx(item->item_code, user->inventory)) != -1) {
        user->inventory[idx]->count += item->count;
        user->coin -= item->cost;
        return true;
    }
    else {
        //해당 아이템이 존재하지 않는경우
        for (int i = 0; i < INVENSIZE; i++) {
            if (user->inventory[i]->item_type == ITEM_TYPE::I_NONE) {
                copy_item(item, user->inventory[i]);
                user->coin -= item->cost;
                return true;
            }
        }
    }
}
```

상점에서 아이템을 살때 사용하는 코드입니다.
아이템별 고유 아이템 코드로 각 아이템을 구분합니다.

만약 아이템이 인벤토리에 있으면 해당 아이템의 개수만 늘려줍니다.

인벤토리를 순회하여 인벤토리에 해당 아이템이 존재하지 않으면 추가해 줍니다.

구현한 기능 설명

(4.2 아이템)

```
void copy_item(Item* base_item, Item* copy_item)
{
    copy_item->cost = base_item->cost;
    copy_item->count = base_item->count;
    copy_item->hp_recovery = base_item->hp_recovery;
    copy_item->item_type = base_item->item_type;
    strcpy(copy_item->description, base_item->description);
    strcpy(copy_item->item_code, base_item->item_code);
}
```

base_item포인터의 값을 copy_item포인터의 값에 복사해주는 코드입니다. 이때 두 변수는 포인터 이므로 Item객체의 변수는 깊은 복사를 해주어야 합니다. 그리고 Item의 변수 Description, item_code는 char*이므로 strcpy(...)를 사용해 메모리를 복사해 줍니다.

구현한 기능 설명

(4.3 아이템)

```
bool use_item(const char* use_item, PlayerCharacter* user, MapInfo* map_info,
float elapsed_time)
{
    int idx = find_item_idx(use_item, user->inventory);
    if (idx == -1 || user->inventory[idx]->count <= 0) return false;

    if (strcmp(use_item, "Item_HPPotion") == 0) {
        user->health = clamp(user->health + user->inventory[idx]->hp_recovery, 0,
        user->max_health);
        user->inventory[idx]->count--;
    }
    else if (strcmp(use_item, "Item_Increase_Max_HP") == 0) {
        user->max_health++;
        user->health = user->max_health;
        user->inventory[idx]->count--;
    }
    else if (strcmp(use_item, "Item_AStar") == 0) {
        AStar(map_info, user->pos[0], user->pos[1], map_info->map_size - 1, map_info->map_size - 1);
        follow_AStar_route(map_info, user->pos[0], user->pos[1], map_info->map_size - 1, map_info->map_size - 1, user, elapsed_time);
        user->pos[0] = map_info->map_size - 1;
        user->pos[1] = map_info->map_size - 1;
        user->inventory[idx]->count--;
    }
    return true;
}
```

해당 함수는 위의 make_item(...)함수와 유사하게
아이템의 기능을 만드는 함수 입니다. 해당
함수에서 아이템의 기능을 정의해야 합니다.

현재 만든 아이템

체력 물약(체력1 증가)

최대체력 증가(최대체력 1 증가)

자동 보물 찾기(A*알고리즘 사용)

구현한 기능 설명

(5.0 메인 화면)

- 메인 함수가 실행되면 핵심으로 `async_loading_and_UI(...)` 함수가 실행된 후 `main_UI(...)`가 실행됩니다.

```
void async_loading_and_UI(BaseInfo* base_info, MapInfo** map_infos, PlayerCharacter* player_info)
{
    Std::future<bool> async_initialize = std::async(std::launch::async, initialize_BMC, base_info, map_infos, player_info);
    Int i = 0;
    While (true) {
        Std::chrono::milliseconds span(300);
        Std::future_status stat = async_initialize.wait_for(span);

        switch (i % 3)
        {
        case 0:
            system("cls");
            display_loading_window1();
            break;
        case 1:
            system("cls");
            display_loading_window2();
            break;
        case 2:
            system("cls");
            display_loading_window3();
            break;
        default:
            break;
        }
        i++;

        if (stat == std::future_status::ready) {
            return;
        }
    }
}
```

Future라는 객체를 사용하여 loading창과 initialize_BMC(...)의 함수 실행을 비동기로 처리 했습니다.

구현한 기능 설명

(5.1 메인 화면)

```
void main_UI(MapInfo** map_infos, PlayerCharacter* user, struct BaseInfo* base_info, Item**
items)
{
    MENU_LIST list = change_main_menu(base_info);

    if (list == MENU_LIST::PLAY) {
        bool isClear = false;
        //미션 실패
        while ((isClear = play_game(user, map_infos)) == false) {
            ... //맵 초기화(reset_map(...)) 및 실패화면 출력
            int input = 0;
            while ((input = getch()) != 'y' && input != 'n');

            if (input == 'n') break;
        }

        //미션 성공
        if (isClear) {
            ... //맵 초기화(reset_map(...))
            system("cls");
            display_curscore(map_infos);
        }
    }
    else if (list == MENU_LIST::STORE) {
        char idx = 0, input = 0;
        bool is_input_enter = false, is_buying = false;
        while ((input = user_input()) != 'q' ) {
            if (input == 72) {
                idx--;
                if (idx < 0) idx = ALLITEMSIZE - 1;
            }

```

```
            else if (input == 80) {
                idx++;
                if (idx == ALLITEMSIZE) idx = 0;
            }
            else if (input == 13) {
                is_input_enter = true;
                is_buying = buy_item(items[idx], user);
                //사자마자 적용될 아이템
                char item_code[] = "Item_Increase_Max_HP";
                if (strcmp(item_code, items[idx]->item_code) == 0) use_item(item_code, user, map_infos[0], 0);
            }

            system("cls");
            printf("보유 코인 수 : %d\n\n", user->coin);
            printf("구매 하려면 엔터\n\n");
            display_store(items, user, idx);
            if (is_input_enter) {
                if (is_buying) {
                    printf("구매완료\n");
                }
                else {
                    printf("코인이 부족합니다\n");
                }
            }
            printf("\n종료하려면 q를 입력하시오");
        }
    }
    else if (list == MENU_LIST::EXIT) return;

    main_UI(map_infos, user, base_info, items);
}
```

구현한 기능 설명

(5.2 메인 화면)

```
MENU_LIST change_main_menu(BaseInfo* base_info)
{
    system("cls");
    display_main_menu(base_info->menu_list);

    int input = 0;
    while (input = user_input()) {
        if (input == 'q') return MENU_LIST::EXIT;
        if (input == 72 || input == 80 || input == 13) break;
    }

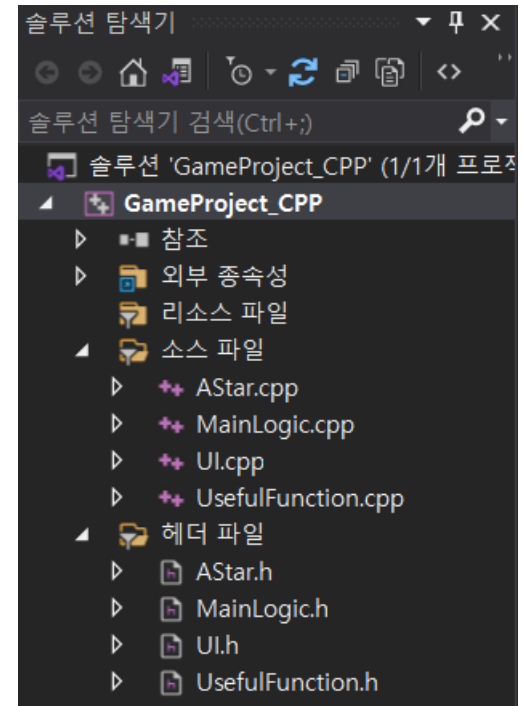
    switch (input)
    {
        case 72: //상
            base_info->menu_list = (MENU_LIST)((base_info->menu_list - 1));
            if (base_info->menu_list == 0) base_info->menu_list = MENU_LIST::STORE;
            return MENU_LIST::M_NONE;
    }
```

```
case 80: //하
    if (base_info->menu_list == base_info->menu_list_size - 1) base_info->menu_list
    = MENU_LIST::PLAY;
    else base_info->menu_list = (MENU_LIST)((base_info->menu_list + 1));
    return MENU_LIST::M_NONE;
}
case 13:
    if (base_info->menu_list == MENU_LIST::PLAY) {
        return MENU_LIST::PLAY;
    }
    else if (base_info->menu_list == MENU_LIST::STORE) {
        return MENU_LIST::STORE;
    }
    default:
        break;
}
```

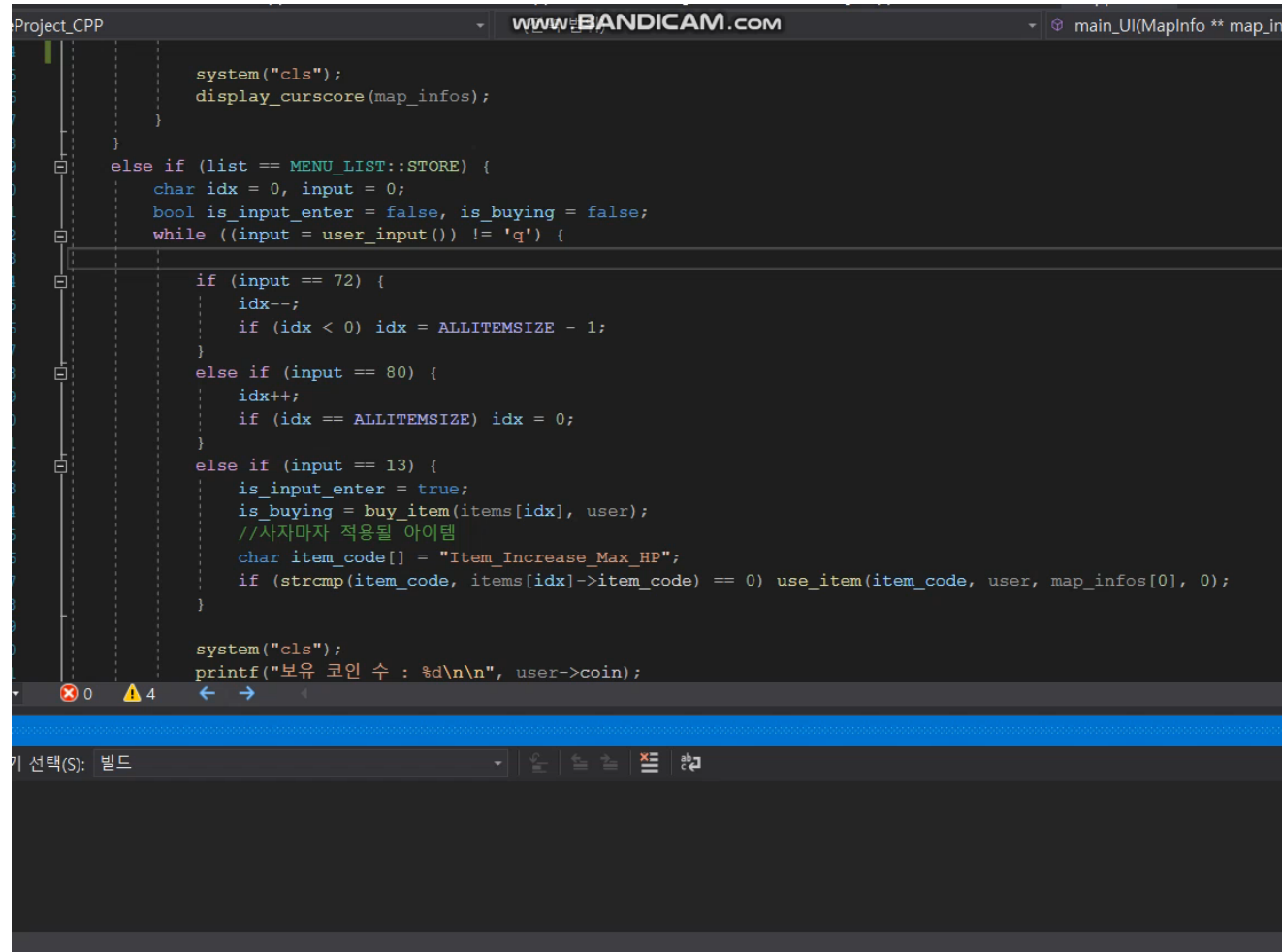
메인화면을 전시해 주고 키입력을 받아 base->info->menu_list의 enum값을 변경 후 리턴해 주는 함수 입니다.

게임 외 코드관리

- 소스파일의 순환 참조를 피하기 위해 헤더파일을 만들어 관리(헤더부분이 서로 참조하면 오류)
- 자주 사용할 것 같은 함수는 파일을 따로 관리
- `const *`를 함수 파라미터에 활용해 불필요한 메모리 복사x, 수정불가
- 함수 파라미터에 포인터 선언 시 함수 초반 널 체크
- 매크로, 열거형
- 오버로딩
- 상속



플레이 영상



```
Project_CPP www.BANDICAM.COM main_UI(MapInfo ** map_in

    system("cls");
    display_curscore(map_infos);
}

else if (list == MENU_LIST::STORE) {
    char idx = 0, input = 0;
    bool is_input_enter = false, is_buying = false;
    while ((input = user_input()) != 'q') {

        if (input == 72) {
            idx--;
            if (idx < 0) idx = ALLITEMSIZE - 1;
        }
        else if (input == 80) {
            idx++;
            if (idx == ALLITEMSIZE) idx = 0;
        }
        else if (input == 13) {
            is_input_enter = true;
            is_buying = buy_item(items[idx], user);
            //사자마자 적용될 아이템
            char item_code[] = "Item_Increase_Max_HP";
            if (strcmp(item_code, items[idx]->item_code) == 0) use_item(item_code, user, map_infos[0], 0);
        }

        system("cls");
        printf("보유 코인 수 : %d\n\n", user->coin);
    }
}
```

<https://youtu.be/mf5FUtNkAFk>

참고 자료

- A*알고리즘 : <https://kangworld.tistory.com/61>
- 비동기 함수 실행 : <https://cclient.tistory.com/m/18>
- Page24 ※ : <https://blog.naver.com/PostView.naver?blogId=pk3152&logNo=221558157798>
- kbhit() : <https://m.blog.naver.com/sharonichoya/220875372940>
- 콘솔창 크기 설정 : <http://egloos.zum.com/EireneHue/v/350607>