**1942 프로젝트**

**중간 보고서**

2학년 3반

1705095 이창민

2022.05.02 제출

**목차**

리스트에 링크를 포함했습니다.

|  |  |  |
| --- | --- | --- |
| **메인화면**   1. [**main.c ( 3~5)**](#main) 2. [**login.c (6~7)**](#login) 3. [**draw\_content.c (8~10)**](#draw_content) 4. [**move.c (11~12)**](#move) 5. [**gotoxy.c (13)**](#gotoxy) 6. [**cursor.c (13)**](#cursor) 7. [**common.h (14)**](#common) | 1. **폭탄기능** 2. [**bomb\_bul\_move.c (29)**](#bomb_bull_move) 3. [**bomb\_bul\_status.c (30~31)**](#bomb_bull_status) 4. [**bomb\_move.c (32~35)**](#bomb_move) 5. [**bomb\_status.c (36~37)**](#bomb_status) | |
| **게임화면**   1. **공통** 2. [**game.c (15~21)**](#game) 3. [**draw\_stat.c (22)**](#draw_stat) 4. [**draw\_symbol.c (23)**](#draw_symbol) 5. [**set\_color.c (23)**](#set_color) 6. [**game.h (24~26)**](#game_h) 7. **플레이어** 8. [**bullet\_move.c (27)**](#Bullet_move) 9. [**bullet\_status.c (28)**](#bullet_status) | | 1. **아이템** 2. [**item\_gen.c (38)**](#item_gen) 3. [**item\_move.c (39)**](#item_move) 4. [**item\_status.c (40~41)**](#item_status) 5. **적** 6. [**enemy\_gen.c (42~44)**](#enemy_gen) 7. [**enemy\_move.c (45~48)**](#enemy_move) 8. [**enm\_bull\_move.c ( 49 ~50)**](#enm_bull_move) 9. [**enm\_bull\_status.c (51~55)**](#enm_bull_status) 10. [**enm\_stauts.c (56~59)**](#enm_status) |
|  | |  |

main.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, printf, puts 사용

#include <stdlib.h> // 표준 라이브러리, system 사용

#include <conio.h> // 콘솔 입출력 함수를 제공, getch사용

#include "common.h" // game.c에서만 사용하는 함수들을 제외한 모든 함수 선언들과 키보드 값을 저장하고 있는 enum을 작성함

#define POS\_X 59 // 매크로 상수, 메인화면에서 사용자의 키보드 X 값을 저장하기 위해 사용

int main(void) // int 형을 반환하고 매개변수가 없는 main 함수

{

    system("mode con: cols=100 lines=30"); // system 함수는 콘솔창 명령어를 입력할 수 있다. 콘솔창 화면 크기를 조절하기 위해 작성

    system("title 1705095 이창민 - 1942 프로젝트"); // 콘솔창의 타이틀을 변경하기 위해 사용

    cursor(0); // 콘솔창에서 깜빡이는 커서를 숨기기 위해 cursor 함수로 작성후 호출함

    int login\_chk = 0; // 로그인 상태를 구별하기 위해 사용하는 int형 변수

    int menu\_num = 18; // 사용자가 선택한 메뉴 (좌표 Y값)을 저장하는 int형 변수

    int key = 0; // getch 로 입력받은 키 값을 저장하기 위해 사용

    while (menu\_num != 24) // 게임종료 (y값 24) 선택전까지 무한정 반복

    {

        system("cls"); // 콘솔창의 화면을 지워주는 명령어

        draw\_content(0); // UI를 그리기 위한 함수, int형 매개변수를 통해 호출 화면을 구분하여 출력한다.

        switch (login\_chk) // 로그인 유무를 구별하기 위한 switch문, 로그인과 비로그인 상태에 따라 화면에 출력하는 항목들이 달라진다.

        {

        default: // 로그인(1) 상태가 아니라면 전부 비로그인으로 구별한다.

            gotoxy(39, 18); // 콘솔창의 x,y 좌표 이동을 위해 선언한 gotoxy 함수

            puts("로그인"); // print문으로 화면에 항목들을 출력

            gotoxy(39, 20);

            puts("회원가입");

            gotoxy(39, 22);

            puts("랭킹화면");

            gotoxy(39, 24);

            puts("게임종료");

            break; // switch 문 탈출을 위한 break

        case 1: // 값이 1이면 로그인 상태로 구별한다.

            gotoxy(39, 18); // print문으로 화면에 항목들을 출력

            printf("게임시작");

            gotoxy(39, 20);

            puts("상점");

            gotoxy(39, 22);

            puts("랭킹확인");

            gotoxy(39, 24);

            puts("게임종료");

            break; // switch 문 탈출을 위한 break

        }

        // 사용자가 메뉴 목록을 선택하기전까지 계속 반복함

        // 입력받아 저장한 key 값이 common.h 헤더파일에 선언한 열거형의 ENTER (13)이 아니라면 계속 반복한다.

        // ENTER를 입력한다면 반복문을 종료한다.

        while (key != ENTER)

        {

            gotoxy(POS\_X, menu\_num); // 콘솔창의 좌표 이동을 위해 사용, X값은 고정이기에 define 매크로 상수를 이용했다.

            puts("◀"); // 현재 선택중인 메뉴를 구별해주기 위해 출력

            key = getch(); // \_getch 함수는 사용자에게 키보드로 하나의 키를 입력받는 함수이고, 입력받은 키를 변수 key에 저장

            gotoxy(POS\_X, menu\_num); // 콘솔창의 좌표 이동을 위해 사용, X값은 고정이기에 define 매크로 상수를 이용했다.

            puts("  "); // ENTER를 입력했다면 다음 화면으로 넘어가기에 문제없지만 방향키를 입력했다면 그전 자리의 메뉴 선택기를 지워주기 위해 빈공간으로 작성함

            // 입력받은 키 값을 기준으로 좌표값을 변경하기 위한 함수, 차례대로 화면번호, 입력받은 키 값, X 값, Y 값을 move 함수 매개변수로 넘겨준다.

            // 다른 화면에서도 사용하기 위해 다양한 값들을 넘겨주었고 메인화면에서는 Y 값만 이동하기 때문에 menu\_num 변수로 정수값을 리턴한다.

            menu\_num = move(0, key, POS\_X, menu\_num);

        }

key = 0; // 메뉴가 선택되면 Key 값은 0으로 초기화한다.

// 0으로 초기화를 안해주면 가장 바깥족 while문이 다시 반복할 때 key 값에는 이미 ENTER 값이 들어 있기 때문에 사용자가 메뉴를 선택할 수 없다.

        // 사용자가 선택한 메뉴 번호를 기준으로 다음 화면을 선택한다.

        switch (menu\_num)

        {

        case 18: // 메뉴목록으로는 첫번째 항목, Y값이 18이어서 18로 구별한다.

            // login\_chk 변수를 2로 나눠서 나머지 값을 보고 로그인 상태인지 비로그인 상태인지 구별한다.

            // 나머지 값이 0이라면 비로그인 상태로 보고 login 함수를 호출하여 로그인 화면으로 넘긴다.

            // 나머지 값이 1이라면 로그인 상태로 보고 game 함수를 호출한다.

            login\_chk % 2 == 0 ? login\_chk = login(1) : game();

            system("mode con: cols=100 lines=30"); // 게임 화면에서는 콘솔창의 크기가 달라지기에 메인화면으로 복귀하면 콘솔창 크기를 바꿔준다.

            break; // switch 문 탈출을 위한 break

        case 20: // 두번째 메뉴 항목

            // 나머지 값이 0이라면 비로그인 상태로 보고 login 함수를 호출하여 회원가입 화면으로 넘긴다.

            // 나머지 갑이 1이라면 로그인 상태로 보고 shop 함수를 호출하여 상점 화면으로 넘긴다.

// 아직 shop.c는 없기 때문에 코드는 주석처리 한다.

            // login\_chk % 2 == 0 ? login\_chk = login(0) : shop();

            break; // switch 문 탈출을 위한 break

        case 22:

            // 나머지 값이 0이라면 ranking 함수를 호출하여 비로그인 상태의 랭킹 화면으로 넘긴다.

            // 나머지 갑이 1이라면 로그인 상태의 랭킹 화면으로 넘긴다.

            // ranking 함수의 매개변수로 정수값 0, 1을 넘겨서 구별한다.

// 아직 ranking.c는 없기 때문에 코드는 주석처리 한다.

            // login\_chk % 2 == 0 ? ranking(0) : ranking(1);

            break; // switch 문 탈출을 위한 break

        case 24: // 게임 종료 항목

            break; // switch 문 탈출을 위한 break

        }

    }

    return 0; // main 함수의 리턴값으로 0을 넘기고 프로그램을 종료한다.

}

login.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, printf사용

#include <stdlib.h> // 표준 라이브러리, malloc, free 사용

#include "common.h" // cursor, gotoxy 함수를 사용하기 위해 작성

int login(int screen\_num) // int형을 리턴하는 login 함수, 전달받는 매개변수로 로그인 화면인지 회원가입 화면인지 구분한다.

{

// user\_id, user\_password 포인터 변수는 임시로 사용중

    char\* user\_id = malloc(sizeof(char) \* 20); // 문자열을 입력받기 위해 char형 포인터 변수 user\_id에 char 사이즈 \* 20만큼의 공간을 할당한다.

    char\* user\_password = malloc(sizeof(char) \* 20); // 문자열을 입력받기 위해 char형 포인터 변수 user\_password에 char 사이즈 \* 20만큼의 공간을 할당한다.

    switch (screen\_num) // switch문을 사용하여 회원가입/로그인 화면 구분

    {

// screen\_num이 0 이라면 기존의 메뉴 항목을 지우고 회원가입 출력을 해준다.

    case 0:

        gotoxy(39, 18);

        printf("                            ");

        gotoxy(39, 20);

        printf("                            ");

        gotoxy(39, 22);

        printf("                            ");

        gotoxy(39, 24);

        printf("                            ");

        cursor(1); // 사용자가 입력하는 순간에는 cursor(1)을 호출하여 콘솔창에 커서가 깜빡이게 만듬

        gotoxy(39, 18);

        printf("회원가입");

        gotoxy(39, 20);

        printf("20자 이내에 입력하십시오");

        gotoxy(39, 22);

        printf("아이디: ");

        scanf("%s", user\_id);

        gotoxy(39, 24);

        printf("비밀번호: ");

        scanf("%s", user\_password); // 입력받은 문자열을 저장한다.

        cursor(0); // 입력이 끝나면 커서를 숨김

// break문이 없는 이유는 회원가입 이후 곧바로 로그인 화면을 보여주기 위함

    case 1: // 1의 경우 로그인 항목을 출력해준다.

        gotoxy(39, 18);

        printf("                            ");

        gotoxy(39, 20);

        printf("                            ");

        gotoxy(39, 22);

        printf("                            ");

        gotoxy(39, 24);

        printf("                            ");

        cursor(1); // 사용자가 입력하는 순간에는 cursor(1)을 호출하여 콘솔창에 커서가 깜빡이게 만듬

        gotoxy(39, 18);

        printf("로그인");

        gotoxy(39, 20);

        printf("아이디: ");

        scanf("%s", user\_id); // 입력받은 문자열을 저장한다.

        gotoxy(39, 22);

        printf("비밀번호: ");

        scanf("%s", user\_password); // 입력받은 문자열을 저장한다.

        cursor(0); // 입력이 끝나면 커서를 숨김

        break; // switch문을 탈출한다.

    }

    free(user\_id); // 메모리 해제를 해준다.

    free(user\_password); // 메모리 해제를 해준다.

    return 1; // 회원가입을 하더라도 로그인을 다시 하기 때문에 로그인 상태임을 리턴함

}

// DB 연동 이후 회원가입, 로그인 코드를 수정해야함

// 위 코드는 임시로 사용

draw\_content.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, printf, puts 사용

#include <stdlib.h> // 표준 라이브러리, system 사용

#include "common.h" // gotoxy 함수를 사용하기 위해 작성

// 리턴 타입 void, 매개변수로 호출하는 시점의 화면 값을 전달 받는 함수

// UI를 그려주는 역할을 함

void draw\_content(int screen\_num)

{

    switch (screen\_num) // switch문을 사용하여 화면 값에 따라 그려줌

    {

    // 메인 화면

    case 0:

        system("cls"); // UI를 그리기전 화면을 한번 지워준다.

        // 메인화면 그림의 숫자 1

        gotoxy(25, 3);

        puts("▥▥");

        gotoxy(23, 4);

        puts("▨▥▥");

        gotoxy(21, 5);

        puts("▨▨▥▥");

        gotoxy(25, 6);

        puts("▥▥");

        gotoxy(25, 7);

        puts("▥▥");

        gotoxy(25, 8);

        puts("▥▥");

        gotoxy(25, 9);

        puts("▥▥");

        gotoxy(21, 10);

        puts("▤▤▥▥▤▤");

        gotoxy(21, 11);

        puts("▤▤▥▥▤▤");

        // 메인화면 그림의 숫자 9

        gotoxy(35, 3);

        puts("▩▤▤▥▥");

        gotoxy(35, 4);

        puts("▥    ▥▥");

        gotoxy(35, 5);

        puts("▥    ▥▥");

        gotoxy(35, 6);

        puts("▩▤▤▥▥");

        gotoxy(41, 7);

        puts("▥▥");

        gotoxy(41, 8);

        puts("▥▥");

        gotoxy(41, 9);

        puts("▥▥");

        gotoxy(41, 10);

        puts("▥▥");

        gotoxy(41, 11);

        puts("▥▥");

        // 메인화면 그림의 숫자 4

        gotoxy(49, 3);

        puts("▥    ▥▥");

        gotoxy(49, 4);

        puts("▥    ▥▥");

        gotoxy(49, 5);

        puts("▥    ▥▥");

        gotoxy(49, 6);

        puts("▥    ▥▥");

        gotoxy(49, 7);

        puts("▩▤▤▥▥▤▤");

        gotoxy(55, 8);

        puts("▥▥");

        gotoxy(55, 9);

        puts("▥▥");

        gotoxy(55, 10);

        puts("▥▥");

        gotoxy(55, 11);

        puts("▥▥");

        // 메인화면 그림의 숫자 2

        gotoxy(65, 3);

        puts("▤▤▤▤▤▥▩");

        gotoxy(65, 4);

        puts("▤▤▤▤▤▥▥");

        gotoxy(75, 5);

        puts("▥▥");

        gotoxy(75, 6);

        puts("▥▥");

        gotoxy(65, 7);

        puts("▩▥▤▤▤▥▥");

        gotoxy(65, 8);

        puts("▥▥▤▤▤▥▩");

        gotoxy(65, 9);

        puts("▥▥");

        gotoxy(65, 10);

        puts("▥▥");

        gotoxy(65, 11);

        puts("▩▥▤▤▤▤▤");

        gotoxy(51, 13);

        puts("1705095 이창민 기말 프로젝트");

        gotoxy(2, 29); // 메인화면 하단의 동작 방법

        puts("조작방법: 방향 ↑ ↓ ← → 선택 ENTER 아이템 Z X");

        break;

    // 게임 화면에서 비행기가 표시되는 구간

    case 1:

        for (int i = 0; i <= 30; i += 1)

        {

            gotoxy(4, i);

            puts(":");

            gotoxy(56, i);

            puts(":");

        }

        break;

    // ESC 키 입력후 PAUSE 상태의 창을 그려줌

    case 2:

        gotoxy(16, 10);

        puts("PAUSE");

        gotoxy(15, 11);

        puts("┌─────────────────────────────┐");

        gotoxy(15, 12);

        puts("│  게임을 종료 하시겠습니까?  │");

        gotoxy(15, 13);

        puts("├─────────────────────────────┤");

        gotoxy(15, 14);

        puts("│   예 (  )  │  아니요 (  )   │");

        gotoxy(15, 15);

        puts("└─────────────────────────────┘");

        break;

    }

}

move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, printf, puts 사용

#include "common.h" // 키보드값을 저장하고 있는 열거형 enum을 사용

// int형 값 (x좌표 또는 y 좌표)을 리턴하며 ‘화면 번호, 사용자가 입력한 값, X좌표, Y좌표’를 전달받는 move 함수

// 화면 번호를 기준으로 x값 또는 y값을 변경하여 리턴한다.

int move(int screen\_num, int key, int POS\_X, int POS\_Y)

{

    switch (screen\_num) // 1. 화면 번호로 무슨 화면인지부터 구별

    {

    // 메인화면

    case 0:

        switch (key) // 전달받은 키 값으로 위, 아래 동작

        {

        case UP: // common.h의 enum에 들어있는 UP

// if문보다 삼항 연산자가 읽기 더 편해서 사용

// 커서가 이미 첫번째 항목을 가리키고 있다면 동작하지 않음

// n 번째 항목이라면 y값 2를 빼준다.

// 2를 빼는 이유는 메뉴의 아래위 간격이 2이기 때문

            POS\_Y = (POS\_Y <= 18) ? 18 : POS\_Y - 2;

            break; // switch문 탈출

        case DOWN: // common.h의 enum에 들어있는 DOWN

// 커서가 마지막 항목을 가리키고 있다면 동작하지 않음

// 마지막 항목이 아니라면 y에 2를 더해준다.

            POS\_Y = (POS\_Y >= 24) ? 24 : POS\_Y + 2;

            break; // switch문 탈출

        }

        return POS\_Y; // Y값만 움직이기 때문에 Y 좌표 리턴

// game.c에서 ESC키 입력후 PAUSE 화면에 들어갔을 때 게임을 종료할지 말지를 묻는 화면에서 사용

    case 1:

        switch (key) // 전달받은 키 값으로 좌, 우 동작

        {

// PAUSE 화면은 예, 아니오 밖에 없어서 x값은 23 아니면 39다.

        case LEFT: // common.h의 enum에 들어있는 LEFT

// 왼쪽 키를 눌렀을 때 X값이 23보다 크면 23을 대입하고 아니라면 그냥 23 그대로 대입한다.

            POS\_X = (POS\_X > 23) ? 23 : POS\_X;

            break; // switch문 탈출

        case RIGHT: // common.h의 enum에 들어있는 RIGHT

// 오른쪽 키를 눌렀을 때 X값이 39보다 작으면 39를을대입하고 아니라면 그냥 39 그대로 대입한다.

            POS\_X = (POS\_X < 39) ? 39 : POS\_X;

            break; // switch문 탈출

        }

        return POS\_X; // X값만 움직이기 때문에 X 좌표 리턴

    }

}

|  |
| --- |
| move.c 함수는 이후 상점, 랭킹 화면에서도 사용할 예정이다.  (상점, 랭킹 화면은 아직 만들지 않았음)  game.c에서 플레이 비행기를 움직일때는 다른 방법을 사용하는데 코드 구조의 문제인지 사용자의 키 입력을 빠르게 처리 못하는 현상이 있어서 사용하지 않았다.  최대한 move 함수를 활용하는 방법으로 코드를 짤려고 했지만 실력 부족으로 시간을 더 지체할 수 없었기에 stackoverflow에 나와있는 방법을 사용하였다. |

gotoxy.c

#include <windows.h> // 커서를 다루는 SetConsoleCursorPosition 함수가 포함되어 있어서 windows.h를 인클루드 한다.

void gotoxy(int x, int y) // 리턴 타입 void, x값과 y값을 전달 받는 gotoxy 함수

{

// COORD는 X 값과 Y값을 멤버로 가지는 구조체 타입이라고 한다.

// 단순하게 X 값과 Y값을 저장한다고 생각하면 된다고 한다.

    COORD pos = { x,y };

    SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), pos);

// 우선 GetStdHandle 함수는 실제 핸들을 반환하는 함수이며 그안에 파라미터로 종류를 정해주면 핸들을 리턴한다.

// ★ 운영체제는 자신의 리소스를 안전하게 관리하기 위해 주소를 사용하는 포인터 대신에 ‘핸들’ 이라는 개념을 사용하는데, ‘핸들’ 은 운영체제 내부에 있는 어떤 리소스의 주소를 정수로 치환한 값이라고 한다.

// STD\_OUTPUT\_HANDLE은 정의를 보면 (unsigned long)-11로 정의되어 있다.

// 즉 GetStdHandle(STD\_OUTPUT\_HANDLE) 한문장은 표준콘솔출력의 핸들을 반환해주는것이다.

// 정리하자면 SetConsoleCursorPositoin 함수에, 콘솔의 출력 핸들을 주고, 위치(pos)를 정해주면 커서가 이동한다는 것이다.

}

cursor.c

#include <windows.h> // 커서를 다루는 SetConsoleCursorPosition 함수가 포함되어 있어서 windows.h를 인클루드 한다.

void cursor(int n) // 리턴 타입 void, int형 값을 전달받는 매개변수 하나를 가지는 cursor 함수

{

    CONSOLE\_CURSOR\_INFO ConsoleCursor; // CONSOLE\_CURSOR\_INFO라는 구조체를 선언

    ConsoleCursor.bVisible = n; // consoleCursor 구조체 안의 bVisible 값을 매개변수로 전달받은 n을 대입, TRUE는 보임, FALSE는 숨김이다.

    ConsoleCursor.dwSize = 1; // 커서의 굵기를 1~100으로 조절할 수 있다.

    SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &ConsoleCursor);

// gotoxy와 같은 동작이다.

// SetConsoleCursorPositoin 함수에, 콘솔의 출력 핸들을 주고, 이번에는 ConsoleCursor 구조체의 주소를 넘겨주어 동작을 한다.

}

common.h

#pragma once // 컴파일러에게 해당 헤더 파일이 한번만 빌드되도록 한다. 표준은 아니라고 한다.

void cursor(int); // cursor 함수 원형 선언

void gotoxy(int, int); // gotoxy 함수 원형 선언

int move(int, int, int, int); // move 함수 원형 선언

void draw\_content(int); // draw\_content 함수 원형 선언

int login(int); // login 함수 원형 선언

void game(void); // game 함수 원형 선언

void setColor(unsigned short, unsigned short); // setColor 함수 원형 선언

enum ControlKey // 키보드 값을 저장하고 있는 열거형

{

// 키보드 방향키 아스키 값을 각각 항목에 저장한다.

    UP = 72, DOWN = 80, ENTER = 13, LEFT = 75, RIGHT = 77, BACKSPACE = 8

};

// 열거형은 명명된 정수형 상수의 집합으로 구성된다.

// 열거형을 선언하면 컴파일러는 열거형 멤버들을 정수형 상수로 인식한다.

// 열거형은 변수를 만들고 그 변수에 특정 멤버변수로 지정하여 사용할 수도 있다.

game.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, printf, puts 사용

#include <stdlib.h> // 표준 라이브러리, system 사용

#include <Windows.h> // GetAsyncKeyState 사용을 위함

#include "common.h" // 모든 화면에서 사용하는 공통적인 함수들을 사용하기 위함

#include "game.h" // 게임 화면과 관련된 함수들과 구조체를 저장

void game(void) // 리턴타입 void, 매개변수가 없는 game 함수

{

    system("mode con: cols=80 lines=32"); // 콘솔창의 크기를 변경한다.

    system("cls"); // 게임 화면을 그리기전 초기화를 해준다.

    // ↓ 플레이어 관련 구조체들 – 플레이어 정보, 폭탄, 스킬 관련

    Player player = { 28, 28, 1, 3, 0 }; // X값, Y값, 생명, 체력, 점수를 저장하는 player 구조체

    Bullet bullet[BULLET\_SIZE] = { 0, }; // X값, Y값, 발사 확인용 값을 저장하는 bullet 구조체 배열, 배열의 사이즈는 BULLET\_SIZE: 4로 할당한다.

    Skill skill = { 3, 0 }; // X키를 입력하면 사용할 수 있는 스킬의 개수, 재사용 시간을 저장하는 구조체

    Item item[ITEM\_SIZE] = { 0, }; // 게임중 드랍되는 아이템의 X값 Y값 드랍 확인, 아이템 타입, 떨어지는 속도를 저장하는 item 구조체 배열, ITEM\_SIZE: 5로 배열크기를 정한다.

// 원작의 폭탄 기능을 보면 비행기 2대가 올라와서 구조체 배열로 만들었다.

// X값, Y값, 폭탄 수량, 속도, 발사 확인

    Bomb bomb[BOMB\_SIZE] = { 29, 24, 1, 5, FALSE, 6, 24, 0, 5, FALSE };

// 마찬가지로 원작의 폭탄 기능을 보면 비행기 2대가 올라와서 일정 시점부터 총알을 발사한다. 그 기능을 구현하기 위해 아래 구조체 배열들을 만들어준다.

// bomb\_bul, bomb\_bul2 로 나뉘는 이유는 비행기가 2대이기 때문

    Bomb\_blt bomb\_bul[BOMB\_BUL\_SIZE] = { 0, }; // 1번 폭탄 총알의 X값 Y값 발사확인 값을 저장한다.

    Bomb\_blt bomb\_bul2[BOMB\_BUL\_SIZE] = { 0, }; // 2번 폭탄 총알의 X값 Y값 발사확인 값을 저장한다.

    // ↓ 적군 관련

    int frame\_count = 0; // while문이 한번 반복할때마다 카운트가 1씩 올라가는데 이를 기준으로 적군을 생성한다.

    Enemy enemy[ENEMY\_SIZE] = { 0, }; // 적군의 X값, Y값, 타입, 움직임 패턴, 움직이는 횟수, 스피드, 체력, 생성 확인, 적군 총알 배열, 적군 총알들의 X값, Y값, 발사 확인, 총알 속도를 저장하는 구조체 배열

    while (1) // 게임을 종료하기전까지 무한 반복

    {

        draw\_content(1); // 비행기가 표시되는 기준을 그려주는 함수

        draw\_stat(player, bomb); // 플레이어의 체력, 폭탄 개수 등을 그리기 위한 함수, player와 bomb 구조체 변수를 전달한다. 배열과 마찬가지로 구초제 변수의 주소는 그 변수의 첫 번째 멤버의 주소값이다.

gotoxy(player.pos\_x, player.pos\_y); // gotoxy 함수에 플레이어의 x,y 값을 전달하여 커서를 이동시킨다.

puts("[-\*-]"); // 단순히 문자열만 출력하는 puts 함수이며 printf보다 빠르다고 한다, 플레이어 비행기 출력

// 반복할때마다 화면을 안 지워주고 일부 항목의 위치만 공백으로 그리고 다시 그려주는 이유는 화면을 지우고 새로 그릴때마다 화면 전체가 깜빡이는 현상이 있기때문

        // VS에서 코드를 접기 위해 블록화를 하였다.

        {

// GetAsyncKeyState함수는 현재 키의 상태를 알아오는 함수이다.

// 이 함수가 호출되기 이전에 키를 누른 상태였는지와 호출된 시점인 현재에도 누르고 있는 상태인지를 알 수 있다.

        // 함수화를 하거나 getch를 사용하면 처리 속도가 매우 느려졌음

// ↓ 왼쪽 방향키가 눌리고 플레이어의 X 값이 5보다 크다면 기존의 플레이어 위치를 공백으로 지워주고 x값을 감소시켜 새로 그려준다.

            if (GetAsyncKeyState(VK\_LEFT) && player.pos\_x > 5)

{

                gotoxy(player.pos\_x, player.pos\_y);

                puts("     ");

                player.pos\_x--;

                gotoxy(player.pos\_x, player.pos\_y);

                puts("[-\*-]");

            }

// ↓ 오른쪽 방향키가 눌리고 플레이어의 X 값이 51보다 작다면 기존의 플레이어 위치를 공백으로 지워주고 x값을 증가시켜 새로 그려준다.

            if (GetAsyncKeyState(VK\_RIGHT) && player.pos\_x < 51)

{

                gotoxy(player.pos\_x, player.pos\_y);

                puts("     ");

                player.pos\_x++;

                gotoxy(player.pos\_x, player.pos\_y);

                puts("[-\*-]");

            }

// ↓ 위쪽 방향키가 눌리고 플레이어의 Y 값이4 보다 크다면 기존의 플레이어 위치를 공백으로 지워주고 Y 값을 감소시켜 새로 그려준다.

            if (GetAsyncKeyState(VK\_UP) && player.pos\_y > 4)

{

                gotoxy(player.pos\_x, player.pos\_y);

                puts("     ");

                player.pos\_y--;

                gotoxy(player.pos\_x, player.pos\_y);

                puts("[-\*-]");

            }

// ↓ 아래쪽 방향키가 눌리고 플레이어의 Y 값이 29 보다 작다면 기존의 플레이어 위치를 공백으로 지워주고 Y 값을 증가시켜 새로 그려준다.

if (GetAsyncKeyState(VK\_DOWN) && player.pos\_y < 29)

{

                gotoxy(player.pos\_x, player.pos\_y);

                puts("     ");

                player.pos\_y++;

                gotoxy(player.pos\_x, player.pos\_y);

                puts("[-\*-]");

            }

        }

        // ESC키가 눌리면 게임을 일시 정지한다.

        if ((GetAsyncKeyState(VK\_ESCAPE)))

        {

            int menu\_num = 23; // 예, 아니요를 구별하는 X 값

            int key = 0; // 키 값을 저장하는 변수

            while (key != ENTER) // 엔터를 입력하기 전까지 무한 반복

            {

                draw\_content(2); // PAUSE 화면을 그려준다.

                gotoxy(menu\_num, 14);

                puts("◎");

                key = getch();

                gotoxy(menu\_num, 14);

                puts("  ");

// 사용자가 왼쪽이나 오른쪽으로 움직인다면 기존의 위치를 공백으로 지워주고 새로 그려준다.

                menu\_num = move(1, key, menu\_num, 14); // move 함수를 사용해서 X 값을 받아 menu\_num에 저장한다

            }

            if (menu\_num == 23) // 네

            {

                // 게임 스코어 저장을 해줘야함

                break;

            }

            else // menu\_num == 39 : 아니요

            {

// PAUSE창이 게임 화면 중앙에 그려지기에 화면을 지워줘야한다.

                system("cls");

// break를 해버리면 다른 항목들이 움직인채로 새로 그려지기에 부자연스러워서 continue를 해준다.

                continue;

            }

        }

        // 스페이스바를 눌렀을 때 플레이어 총알 공격

        if ((GetAsyncKeyState(VK\_SPACE)))

        {

// 총알 사이즈 만큼의 for문을 돌려준다.

            for (int i = 0; i < BULLET\_SIZE; i++)

            {

// 총알 각각의 발사 유무를 검사하여 발사하지 않는다면 발사한다.

                if (bullet[i].con == FALSE)

                {

// 총알의 발사 위치는 플레이어 비행기의 X값과 Y값을 기준으로 한다.

                    bullet[i].pos\_x = player.pos\_x + 1;

                    bullet[i].pos\_y = player.pos\_y - 1;

                    bullet[i].con = TRUE; // 총알의 발사 유무를 TRUE로 바꿔준다.

                    break; // for문 탈출

                }

            }

        }

// Z키가 눌리고 while문을 한번이라도 돌았을 때 발사

// frame\_count가 1 이상인지 확인하는 이유는 GetAsyncKeyState가 호출하기 이전에 눌렸던 키도 인식하기 때문에 로그인 화면에서 z가 눌린다면 시작과 동시에 폭탄을 발사 해버린다.

        if (GetAsyncKeyState(0x5A) && frame\_count > 0)

        {

// 폭탄의 개수가 1발 이상이고 폭탄1, 폭탄2 모두 FALSE 상태라면 발사한다.

// 폭탄1, 폭탄2의 발사유무를 확인하는 이유는 폭탄2가 폭탄1보다 느리게 사라지기 때문에 폭탄2가 사라지기도 전에 폭탄을 다시 발사하여 화면에 잔상을 남기는 현상을 방지하기 위함이다.

            if (bomb[0].count > 0 && (bomb[0].con == FALSE && bomb[1].con == FALSE))

            {

                bomb[0].count--; // 폭탄 수량 차감

                bomb[0].con = TRUE; // 폭탄 발사

                bomb[1].con = TRUE; // 폭탄 발사

            }

        }

        // 스킬 X키 - 생명 추가, 추후 상점에서 구매하는 방식으로 변경할 예정

        // 추후 STAT UI에 스킬 개수를 표시해줘야함

        // X키가 눌리고 한번 이상 반복, 스킬 개수가 1개 이상, 재사용 시간이 0일 때 스킬 사용

        if ((GetAsyncKeyState(0x58) && frame\_count > 1) && (skill.life\_count == 0 && skill.life\_plus > 0))

        {

            player.life++; // 스킬의 기능인 생명 추가

            skill.life\_plus--; // 스킬 수량 감소

            skill.life\_count = 1000; // 스킬 재사용 시간 할당

// 스킬을 사용하면 폭탄도 1회 무료 발사

            if (bomb[0].count > 0 && (bomb[0].con == FALSE && bomb[1].con == FALSE))

            {

                bomb[0].con = TRUE; // 폭탄 발사

                bomb[1].con = TRUE; // 폭탄 발사

            }

            else

            {

                bomb[0].count++; // 만약 폭탄이 이미 발사중인 상태에서 스킬을 사용했다면 폭탄 수량을 늘려준다.

            }

        }

        // 적군을 생성하는 함수, enemy 구조체와 while문 반복 횟수를 전달한다.

        Enemy\_gen(enemy, frame\_count);

        // 아이템을 생성하는 함수, item 구조체와 while문 반복 횟수를 전달한다.

        itemGen(item, frame\_count);

        // 적군을 이동하는 함수, enemy 구조체를 전달한다.

        enemy\_move(enemy);

        // 폭탄 이동, 폭탄1이나 2가 발사중이라면 동작한다.

        // 폭탄 2가 1보다 느리게 사라지기 때문에 or 검사를 한다.

        if (bomb[0].con == TRUE || bomb[1].con == TRUE)

            bomb\_move(bomb); // 폭탄을 이동시켜주는 함수, bomb 구조체를 전달한다.

        // 총알을 이동시키는 함수, bullet 구조체를 넘긴다.

        bullet\_move(bullet);

// 적군 총알 관련 함수 – 적군 총알 발사, enemy, player, bomb, bullet, 기능 구분용 값을 전달한다.

// 리턴값으로 player 구조체를 넘겨받는데 적군 총알 상태 관련 기능들을 모두 포함하고 있기 때문에 여기서는 필요없지만 일단 넘겨받는다.

        player = enm\_bull\_status(enemy, player, bomb, bullet, 0);

        // 적 총알을 이동시키는 함수, enemy 구조체를 전달한다.

        enm\_bull\_move(enemy);

        // 아이템 이동 함수, item 구조체를 넘긴다.

        item\_move(item);

        // 아이템 상태 함수 – 유저와 충돌 검사

        // item, player, bullet 구조체, 동작 확인용 값을 넘겨준다.

        // bullet 구조체를 넘겨주는 이유는 총알 관련 아이템 때문이다.

        // 나중에 체력, 폭탄 등의 아이템도 추가할 예정이기에 player 구조체도 넘겨준다.

        item\_status(item, player, bullet, 0);

        // 아이템 상태 함수 – 아이템 바닥 충돌

        item\_status(item, player, bullet, 1);

        // 적군 총알 발사 함수 – 적 총알과 플레이어 충돌 검사, enemy, player, bomb, bullet, 기능 구분용 값을 전달한다.

        player = enm\_bull\_status(enemy, player, bomb, bullet, 1);

        // 적군 총알 발사 함수 – 적 총알 바닥 도달 검사

        player = enm\_bull\_status(enemy, player, bomb, bullet, 2);

        // 적군 총알 발사 함수 – 적 총알 플레이어 총알 충돌 검사

        player = enm\_bull\_status(enemy, player, bomb, bullet, 3);

        // 적군 총알 발사 함수 – 폭탄 발사시 적 총알 폭탄 앞에서 삭제

        player = enm\_bull\_status(enemy, player, bomb, bullet, 4);

        // 폭탄 상태 관련 함수 - 폭탄과 적 충돌 검사

        bomb\_status(bomb, enemy, 0);

        // 폭탄 상태 관련 함수 – 폭탄이 천장에 도달하면 삭제

        bomb\_status(bomb, enemy, 1);

        // 적군 상태 관련 함수 - 적군이 바닥에 도달하면 삭제

        // 적군의 상태에 이상을 주는 요소들을 포함하고 있는 구조체들을 넘겨준다.

        // 플레이어의 상태에 변화를 주는 항목이 있어 player 구조체로 리턴받는다.

        player = enm\_status(enemy, bomb, bomb\_bul, bomb\_bul2, player, bullet, 0);

        // 적군 상태 관련 함수 - 플레이어와 적 충돌

        player = enm\_status(enemy, bomb, bomb\_bul, bomb\_bul2, player, bullet, 1);

        // 적군 상태 관련 함수 – 플레이어 총알과 적 충돌

        player = enm\_status(enemy, bomb, bomb\_bul, bomb\_bul2, player, bullet, 2);

        // 적군 상태 관련 함수 - 폭탄 총알과 적 충돌

        player = enm\_status(enemy, bomb, bomb\_bul, bomb\_bul2, player, bullet, 3);

        // 총알 상태 함수 - 총알이 천장에 도달하면 삭제

        // 총알이 적과 충돌하는 것은 적군의 상태에 변동을 주기 때문에 enm\_status 함수에서 처리했음

        bullet\_status(bullet);

        // 폭탄 총알 상태 함수 - 총알 발사

        bomb\_bull\_status(bomb, bomb\_bul, bomb\_bul2, 0);

        // 폭탄 총알 이동 함수

        bomb\_bull\_move(bomb, bomb\_bul, bomb\_bul2);

        // 폭탄 총알 상태 함수 – 폭탄 총알이 천장에 도달하면 삭제

        // 폭탄 총알에 의한 적 파괴는 적군의 상태에 변동을 주기 때문에 enm\_status에서 처리했음

        bomb\_bull\_status(bomb, bomb\_bul, bomb\_bul2, 1);

        // 스킬 재사용 대기시간, while문을 한번 반복할때마다 1씩 차감

        if (skill.life\_count > 0)

            skill.life\_count--;

        // 게임 종료 검사, 플레이어의 체력과 생명이 0이라면 게임 종료

        if (player.life == 0 && player.health == 0)

        {

        // DB 연동후 점수 저장 필요함

            break;

        }

        frame\_count++; // 게임 시간을 측정하는 변수, 적군 생성 기준이 된다.

        Sleep(22); // 콘솔창을 멈춰주는 Sleep 함수, 게임 속도를 조절하기 위해 사용한다.

    }

}

draw\_stat.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, printf, puts 사용

#include "game.h" // 구조체 정보와 draw\_symbol 사용을 위해 포함

#include "common.h" // gotoxy를 사용함

// player, bomb 구조체를 전달 받아 스탯을 그리는 draw\_stat 함수

// 구조체 배열을 매개변수로 넘기면 주소를 넘기기 때문에 포인터로 받아줘야 한다.

void draw\_stat(Player player, Bomb\* bomb)

{

    int pos\_y = 0; // 스탯을 그리기 위한 Y값

    // 생명

    // y값을 5로 할당하고 그려줌

    pos\_y = 5;

    gotoxy(58, pos\_y);

    puts("LIFE");

    draw\_symbol(pos\_y, 1, player.life); // 그려야할 위치와 플레이어의 생명 값을 전달하여 그리는 draw\_symbol 함수

    // draw\_symbol 함수로 만든 이유는 반복적인 작업이기 때문

    // 폭탄

    pos\_y = 8;

    gotoxy(58, pos\_y);

    puts("BOMB");

    draw\_symbol(pos\_y, 2, bomb[0].count); // 폭탄의 남은 개수를 그려줌

    // 체력

    pos\_y = 11;

    gotoxy(58, pos\_y);

    puts("HP");

    draw\_symbol(pos\_y, 3, player.health); // 체력을 그려줌

    // 점수를 출력

    gotoxy(58, 2);

    puts("SCORE");

    gotoxy(58, 3);

    printf("%d", player.score);

}

draw\_symbol.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy 사용을 위해 포함

// Y값과 아이콘의 종류, 개수를 넘겨받는다.

void draw\_symbol(int pos\_y, int symbol, int num)

{

    int pos\_x = 58; // x좌표 설정

// 아이콘을 그리기전 기존의 아이콘 자리를 공백으로 지워줌

    gotoxy(58, pos\_y + 1);

    puts("          ");

// 전달받은 개수만큼 for문을 반복하면서 아이콘을 그려준다.

    for (int i = 0; i < num; i++)

    {

        gotoxy(pos\_x, pos\_y + 1);

        if (symbol == 1)

        puts("\u2665"); // ♥ 그림 유니코드

        else if (symbol == 2)

        puts("\u2299"); // ⊙ 그림 유니코드

        else

        puts("\u2605"); // ★ 그림 유니코드

        pos\_x += 2; // 유니코드는 2칸을 차지하기 때문에 +2를 해줌

    }

}

set\_color.c

#include <Windows.h> // 콘솔창의 색깔을 다루는 SetConsoleTextAttribute 함수가 포함되어 있어서 windows.h를 인클루드 한다.

void set\_color(unsigned short text) // unsigned형 short 값을 전달받는 set\_color 함수

{

    SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), text);

// 정리하자면 SetConsoleTextAttribute 함수에, 콘솔의 출력 핸들을 주고, text 값을주면 출력 색깔이 변한다.

}

// set\_color.c는 상점, 랭킹 화면에서도 사용하기 위해 common.h에 추가한 함수이지만 게임과 더 관련이 있다고 생각하기에 game쪽 설명에 포함함

game.h

#pragma once // 컴파일러에게 해당 헤더 파일이 한번만 빌드되도록 한다. 표준은 아니라고 한다.

// 구조체 배열, TRUE, FALSE를 위해 정의한 매크로 상수들

#define TRUE 1

#define FALSE 0

#define STAT 4

#define BULLET\_SIZE 4

#define BOMB\_SIZE 2

#define BOMB\_BUL\_SIZE 5

#define ITEM\_SIZE 5

#define ENEMY\_SIZE 30

#define ENEMY\_BUL\_SIZE 10

// 구조체: 사용자가 C언어의 기본 타입을 가지고 새롭게 정의할 수 있는 사용자 정의 타입

// typedef: 이미 존재하는 타입에 새로운 이름을 붙일 때 사용한다.

// 구조체 변수를 선언하거나 사용할 때에는 매번 struct 키워드를 사용하여 구조체임을 명시해야 하는데 typedef 키워드를 사용하여 구조체에 이름을 선언하면 편하게 사용할 수 있다

// 구조체를 사용한 이유: 항목별로 저장해야 하는 값중 변수명이 겹치는 항목들이 많아서 한번에 묶어서 저장하기 위해 사용함

// 플레이어 관련 구조체들

typedef struct \_PLAYER

{

    int pos\_x;

    int pos\_y;

    int life;

    int health;

    int score;

} Player; // 플레이어 x,y,생명,체력,점수를 저장한다.

typedef struct \_BULLET

{

    int pos\_x;

    int pos\_y;

    int con;

    int type;

} Bullet; // 플레이어 총알의 x, y, 발사유뮤 확인, 총알 종류를 저장한다.

typedef struct \_SKILL

{

    int life\_plus;

    int life\_count;

} Skill; // 플레이어의 스킬 개수, 재사용 시간을 저장한다.

typedef struct \_BOMB

{

    int pos\_x;

    int pos\_y;

    int count;

    int speed;

    int con;

} Bomb; // 폭탄의 x, y, 개수, 속도, 발사유무를 저장한다.

typedef struct \_BOMB\_BULLET

{

    int pos\_x;

    int pos\_y;

    int con;

} Bomb\_blt; // 폭탄 총알의 x, y, 발사유무를 저장한다.

typedef struct \_ITEM

{

    int pos\_x;

    int pos\_y;

    int con;

    int type;

    int speed;

} Item; // 아이템의 x, y, 드랍 유무, 타입, 속도를 저장한다.

// 적군 관련

typedef struct \_ENEMY

{

    int pos\_x;

    int pos\_y;

    int type;

    int move\_pattern;

    int move\_count;

    int speed;

    int health;

    int con;

    int bullet[10];

    int bul\_pos\_x[10];

    int bul\_pos\_y[10];

    int bul\_con[10];

    int bul\_speed;

} Enemy; // 적군의 x, y, 타입, 이동 패턴, 이동 횟수, 속도, 체력, 생성 유무, 적군 총알 배열, 총알들의 x, y, 발사유무, 발사 빈도 속도를 저장한다.

// 게임 화면 관련 함수 원형 선언들

// 함수 원형에서 구조체를 작성했기 때문에 구조체 정의 아래에 작성해야 한다.

void draw\_symbol(int, int, int);

void draw\_stat(Player, Bomb\*);

void item\_gen(Item\*, int);

void item\_move(Item\*);

void item\_status(Item\*, Player, Bullet\*, int);

void enemy\_gen(Enemy\*, int);

void enemy\_move(Enemy\*);

Player enm\_status(Enemy\*, Bomb\*, Bomb\_blt\*, Bomb\_blt\*, Player, Bullet\*, int);

void enm\_bull\_move(Enemy\*);

Player enm\_bull\_status(Enemy\*, Player, Bomb\*, Bullet\*, int);

void bomb\_move(Bomb\*);

void bullet\_status(Bullet\*);

void bullet\_move(Bullet\*);

void bomb\_move(Bullet\*);

void bomb\_bull\_move(Bomb\*, Bomb\_blt\*, Bomb\_blt\*);

void bomb\_status(Bomb\*, Enemy\*, int);

void bomb\_bull\_status(Bomb\*, Bomb\_blt\*, Bomb\_blt\*, int);

|  |
| --- |
| 상점 기능, 점수, 적군 타입 추가, 타입별 움직임 등을 추가할 때 구조체 관련으로 수정하거나 추가할 부분들이 많다. |

bullet\_move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy 사용을 위해 포함

#include "game.h" // bullet 구조체 사용을 위해 포함

// 총알의 움직임을 관리한다.

void bullet\_move(Bullet\* bullet) // bullet 구조체를 매개변수로 하는 함수

{

// 총알 개수만큼 for문으로 반복한다.

    for (int i = 0; i < BULLET\_SIZE; i++)

    {

// 총알 상태가 발사 상태라면 이동시킨다.

        if (bullet[i].con == TRUE)

        {

// 원래 총알이 있던 위치를 공백으로 지워준다.

            gotoxy(bullet[i].pos\_x, bullet[i].pos\_y);

            puts("  ");

// 총알은 아래에서 위로 향하기 때문에 y값 차감

            bullet[i].pos\_y--;

// 총알의 타입별로 출력해준다.

            gotoxy(bullet[i].pos\_x, bullet[i].pos\_y);

            switch (bullet[i].type)

            {

            case 0:

                puts("ⅰ");

                break;

            case 1:

                puts("ⅱ");

                break;

            case 2:

                puts("ⅲ");

                break;

            }

        }

    }

}

|  |
| --- |
| 아이템 종류와 점수 부분을 구현한뒤 총알 타입별 데미지 부분을 추가해줘야 한다. |

bullet\_status.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // TRUE, FALSE 사용을 위해 포함

#include "game.h" // bullet 구조체 사용을 위해 포함

// bullet 구조체를 매개변수로 하는 함수

// 총알이 천장에 도달하면 삭제한다.

void bullet\_status(Bullet\* bullet)

{

// 총알 수량 만큼 for문을 반복한다.

    for (int i = 0; i < BULLET\_SIZE; i++)

    {

// 총알이 발사 상태이고 Y값이 0 (천장) 이라면

        if (bullet[i].con == TRUE && bullet[i].pos\_y == 0)

        {

// 총알 위치를 공백으로 지우고

            gotoxy(bullet[i].pos\_x, bullet[i].pos\_y);

            puts("  ");

// 발사 상태를 FALSE로 바꾼다.

            bullet[i].con = FALSE;

        }

    }

}

|  |
| --- |
| 공백으로 기존 위치를 지우더라도 그 자리로 다른 요소가 이동한다면 그려지기 때문에 상관 없음 – 자연스럽게 그려짐 |

bomb\_bull\_move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // bomb, bomb\_bul, bomb\_bul2 구조체 사용을 위해 포함

// bomb, bomb\_bul, bomb\_bul2 구조체를 매개변수로 받는 함수

// 폭탄 총알의 움직임을 관리한다.

void bomb\_bull\_move(Bomb\* bomb, Bomb\_blt\* bomb\_bul, Bomb\_blt\* bomb\_bul2)

{

    // 폭탄의 총알 개수만큼 반복

for (int i = 0; i < BOMB\_BUL\_SIZE; i++)

    {

        // 1번 폭탄의 i번째 총알이 TRUE라면

        if (bomb\_bul[i].con == TRUE)

        {

// 기존 위치의 총알을 공백으로 지워준다.

            gotoxy(bomb\_bul[i].pos\_x, bomb\_bul[i].pos\_y);

            puts("                      ");

            bomb\_bul[i].pos\_y--; // 총알의 y값을 차감한다.

// 바뀐 위치로 총알을 그려준다.

            gotoxy(bomb\_bul[i].pos\_x, bomb\_bul[i].pos\_y);

            puts("    ㅆ              ㅆ");

        }

        // 2번 폭탄 또한 1번 폭탄과 동일한 동작을 한다.

        if (bomb\_bul2[i].con == TRUE)

        {

            gotoxy(bomb\_bul2[i].pos\_x, bomb\_bul2[i].pos\_y);

            puts("                      ");

            bomb\_bul2[i].pos\_y--;

            gotoxy(bomb\_bul2[i].pos\_x, bomb\_bul2[i].pos\_y);

            puts("    ㅆ              ㅆ");

        }

    }

}

bomb\_bull\_status.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // bomb, bomb\_bul, bomb\_bul2 구조체 사용을 위해 포함

// bomb, bomb\_bul, bomb\_bul2 구조체, 호출 시점을 구분하기 위한 int형 변수를 매개변수로 하는 함수

// 폭탄 총알의 상태를 관리한다.

void bomb\_bull\_status(Bomb\* bomb, Bomb\_blt\* bomb\_bul, Bomb\_blt\* bomb\_bul2, int num)

{

    switch (num) // num을 기준으로 하는 switch문

    {

    // 폭탄 총알 발사

    case 0:

        // 폭탄 총알 개수 만큼 for문을 반복한다.

        for (int i = 0; i < BOMB\_BUL\_SIZE; i++)

        {

            // 폭탄 총알의 시작 위치가 다른 이유는 2번 폭탄이 1번 폭탄보다 늦게 출력되기 때문이다.

            // 1번 폭탄의 Y값이 13 ~ 18일 때 폭탄의 총알을 발사한다.

            if ((bomb[0].pos\_y >= 13 && bomb[0].pos\_y <= 18) && bomb\_bul[i].con == FALSE)

            {

                bomb\_bul[i].pos\_x = bomb[0].pos\_x; // 폭탄 총알의 x 값은 폭탄의 x 값을 대입

                bomb\_bul[i].pos\_y = bomb[0].pos\_y - 2; // 폭탄 총알의 y 값은 폭탄보다 2칸위를 대입

                bomb\_bul[i].con = TRUE; // 폭탄 총알 발사 유무를 TRUE 값으로 대입

            }

            // 2번 폭탄의 Y값이 14 ~ 19일 때 폭탄의 총알을 발사한다.

            if ((bomb[1].pos\_y >= 14 && bomb[1].pos\_y <= 19) && bomb\_bul2[i].con == FALSE)

            {

            // 2번 폭탄의 x값과 y값을 할당하고 발사 유무를 TRUE로 변경

                bomb\_bul2[i].pos\_x = bomb[1].pos\_x;

                bomb\_bul2[i].pos\_y = bomb[1].pos\_y - 2;

                bomb\_bul2[i].con = TRUE;

            }

        }

        break; // switch문 탈출

    // 폭탄 총알이 천장 도달

    case 1:

        // 폭탄 총알 개수 만큼 for문을 반복한다.

        for (int i = 0; i < BOMB\_BUL\_SIZE; i++)

        {

            // 폭탄의 총알들이 천장에 도달하면 총알을 삭제한다.

            // 폭탄의 총알이 발사 상태이며 y값이 0이라면

            if (bomb\_bul[i].con == TRUE && bomb\_bul[i].pos\_y == 0)

            {

            // 폭탄 총알의 위치를 공백으로 지우고 총알 발사 유무를 FALSE로 한다

                gotoxy(bomb\_bul[i].pos\_x, bomb\_bul[i].pos\_y);

                puts("                          ");

                bomb\_bul[i].con = FALSE;

            }

            // 2번 폭탄의 총알 또한 1번 폭탄과 동일한 동작을 한다.

            if (bomb\_bul2[i].con == TRUE && bomb\_bul2[i].pos\_y == 0)

            {

                gotoxy(bomb\_bul2[i].pos\_x, bomb\_bul2[i].pos\_y);

                puts("                          ");

                bomb\_bul2[i].con = FALSE;

            }

        }

        break; // switch문 탈출

    }

}

bomb\_move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // bomb 구조체 사용을 위해 포함

// bomb 구조체를 매개변수로 하는 함수

// 폭탄의 움직임을 담당한다.

void bomb\_move(Bomb\* bomb)

{

// 아래 변수들은 없어도 상관없지만 코드 길이를 줄이기 위해 사용

    int pos\_x = 0; // 폭탄의 x값을 저장할 int형 변수 선언

    int pos\_y = 0; // 폭탄의 y값을 저장할 int형 변수 선언

    pos\_x = bomb[0].pos\_x; // 1번 폭탄의 X값을 저장

    pos\_y = bomb[0].pos\_y; // 1번 폭탄의 Y값을 저장

    // 1번 폭탄이 TRUE 상태인지 검사한다.

    // 2번 폭탄이 1번 폭탄보다 느리게 끝나기 때문에 검사하지 않으면

    // 2번 폭탄이 끝나기 직전에 화면 하단에 1번 폭탄이 찍혀버린다.

    if (bomb[0].con == TRUE)

    {

        // 원작 게임을 보면 폭탄이 일정 구간까지는 빨리 오다가 천천히 움직이면서 공격을 하고 다시 빨리 없어지는 효과가 있다.

        // 이를 구현하기 위해서 폭탄의 속도를 느리게 움직일때는 20번마다 한번씩 움직이게 하고 빨리 움직일때는 매턴마다 움직이게 한다.

        // Y값이 13이상 18이하일때는 폭탄 1번을 느리게 움직이게 한다.

        if ((pos\_y >= 13 && pos\_y <= 18) && bomb[0].speed < 1)

        {

        // 기존 비행기의 위치를 공백으로 지워준다.

            gotoxy(pos\_x, pos\_y);

            puts("                           ");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("   ");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("           ");

        // 비행기의 y값을 차감시킨다.

            bomb[0].pos\_y--;

            pos\_y = bomb[0].pos\_y;

        // 변경된 위치에서 비행기를 출력해준다.

            gotoxy(pos\_x, pos\_y);

            puts("<<[[\*]]:=:{-\*-}:=:[[\*]]>>");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("| |");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("<[[::=::]]>");

            bomb[0].speed = 20;

        }

        // Y값이 18이상이거나 13이하 일때는 비행기가 빠르게 움직이게 한다.

        if (pos\_y > 18 || pos\_y < 13)

        {

        // 기존 비행기의 위치를 공백으로 지워준다.

            gotoxy(pos\_x, pos\_y);

            puts("                           ");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("   ");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("           ");

        // 비행기의 y값을 차감시킨다.

            bomb[0].pos\_y--;

            pos\_y = bomb[0].pos\_y;

        // 변경된 위치에서 비행기를 출력해준다.

            gotoxy(pos\_x, pos\_y);

            puts("<<[[\*]]:=:{-\*-}:=:[[\*]]>>");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("| |");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("<[[::=::]]>");

        }

    }

    // 1번 폭탄의 y값이 18 이하이거나 1번 폭탄이 FASLE이고 2번 폭탄이 TRUE일 때 움직인다.

    // 첫번째 조건의 이유는 1번 폭탄의 y값이 18일 때 2번 폭탄이 움직이기 때문

    // 두번째 조건은 1번 폭탄이 먼저 사라지고 2번 폭탄만 남았을때도 움직이기 위해 필요하다.

    if (pos\_y < 18 || (bomb[0].con == FALSE && bomb[1].con == TRUE))

    {

        pos\_x = bomb[1].pos\_x; // 2번 폭탄의 x 값 대입

        pos\_y = bomb[1].pos\_y; // 2번 폭탄의 y 값 대입

        // 2번 폭탄이 천천히 움직인다.

        // 1번 폭탄과 동작 방법은 동일하다.

        // 다만, 1번 폭탄과 겹치지 않게 y값의 기준이 다르다.

        if ((pos\_y >= 14 && pos\_y <= 19) && bomb[1].speed < 1)

        {

        // 기존 비행기의 위치를 공백으로 지워준다.

            gotoxy(pos\_x, pos\_y);

            puts("                         ");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("   ");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("           ");

        // 비행기의 y값을 차감시킨다.

            bomb[1].pos\_y--;

            pos\_y = bomb[1].pos\_y;

        // 변경된 위치에서 비행기를 출력해준다.

            gotoxy(pos\_x, pos\_y);

            puts("<<[[\*]]:=:{-\*-}:=:[[\*]]>>");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("| |");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("<[[::=::]]>");

            bomb[1].speed = 20;

        }

        // 2번 폭탄이 등장할때와 사라질 때 빨리 움직인다.

        if (pos\_y > 19 || pos\_y < 14)

        {

        // 기존 비행기의 위치를 공백으로 지워준다.

            gotoxy(pos\_x, pos\_y);

            puts("                         ");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("   ");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("   ");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("           ");

        // 비행기의 y값을 차감시킨다.

            bomb[1].pos\_y--;

            pos\_y = bomb[1].pos\_y;

        // 변경된 위치에서 비행기를 출력해준다.

            gotoxy(pos\_x, pos\_y);

            puts("<<[[\*]]:=:{-\*-}:=:[[\*]]>>");

            gotoxy(pos\_x + 11, pos\_y + 1);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 2);

            puts("| |");

            gotoxy(pos\_x + 11, pos\_y + 3);

            puts("| |");

            gotoxy(pos\_x + 7, pos\_y + 4);

            puts("<[[::=::]]>");

        }

        bomb[1].speed--; // 폭탄의 움직이는 속도를 위해 사용

    }

    bomb[0].speed--; // 폭탄의 움직이는 속도를 위해 사용

}

|  |
| --- |
| 폭탄의 움직임 속도를 조절하는 방식은  특정 위치에서는 20턴에 한번씩 움직이게 하고 빨리 움직일때는 매턴마다 움직이게 해서 조절했다.  20턴을 설정한 이유는 가장 자연스럽게 보였기 때문이다. |

bomb\_status.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // bomb, enemy 구조체 사용을 위해 포함

// bomb, enemy 구조체, 동작 구별용 int 변수를 매개변수로 하는 함수

// 폭탄의 상태를 관리한다.

void bomb\_status(Bomb\* bomb, Enemy\* enemy, int num)

{

    switch (num) // num 값을 기준으로하는 switch문

    {

    // 폭탄과 적이 충돌했을 때

    case 0:

// 폭탄 1이나2가 TRUE 상태인지 검사

        if (bomb[0].con == TRUE || bomb[1].con == TRUE)

        {

            // 폭탄 개수만큼 for문 반복

            for (int i = 0; i < BOMB\_SIZE; i++)

            {

            // 폭탄과 적이 충돌했는지 확인하기 위해 적 비행기 수 만큼 반복

                for (int j = 0; j < ENEMY\_SIZE; j++)

                {

            // 폭탄과 적군의 y값이 동일하거나 겹친다면 적 기체 파괴

                    if (bomb[i].pos\_y == enemy[j].pos\_y || (enemy[j].pos\_y - bomb[i].pos\_y > 0))

                    {

// 적 기제가 있던 자리를 공백으로 지워준다.

                        gotoxy(enemy[j].pos\_x, enemy[j].pos\_y);

                        puts("      ");

                        enemy[j].pos\_x = 0; // 적 기체의 x값 초기화

                        enemy[j].pos\_y = 0; // 적 기체의 y값 초기화

                        enemy[j].con = FALSE; // 적 기체 상태를 FALSE로 변경

                    }

                }

            }

        }

        break; // switch문 탈출

    // 폭탄이 천장에 도달하면 삭제

    // 다만, 폭탄 비행기는 두개이고 진행 속도가 다르기 때문에 각자 처리해준다.

    case 1:

        // 1번 폭탄이 발사 상태이고 y값이 0이라면 삭제한다.

        if (bomb[0].con == TRUE && bomb[0].pos\_y == 0)

        {

            // 1번 폭탄 비행기를 공백으로 지워준다.

            gotoxy(bomb[0].pos\_x, bomb[0].pos\_y);

            puts("                         ");

            gotoxy(bomb[0].pos\_x + 11, bomb[0].pos\_y + 1);

            puts("   ");

            gotoxy(bomb[0].pos\_x + 11, bomb[0].pos\_y + 2);

            puts("   ");

            gotoxy(bomb[0].pos\_x + 11, bomb[0].pos\_y + 3);

            puts("   ");

            gotoxy(bomb[0].pos\_x + 7, bomb[0].pos\_y + 4);

            puts("           ");

            bomb[0].pos\_y = 24; // 1번 폭탄의 y값을 초기값으로 변경

            bomb[0].con = FALSE; // 1번 폭탄의 상태를 FALSE로 변경

        }

        // 1번 폭탄 코드와 동일하다.

        if (bomb[1].con == TRUE && bomb[1].pos\_y == 0)

        {

            gotoxy(bomb[1].pos\_x, bomb[1].pos\_y);

            puts("                         ");

            gotoxy(bomb[1].pos\_x + 11, bomb[1].pos\_y + 1);

            puts("   ");

            gotoxy(bomb[1].pos\_x + 11, bomb[1].pos\_y + 2);

            puts("   ");

            gotoxy(bomb[1].pos\_x + 11, bomb[1].pos\_y + 3);

            puts("   ");

            gotoxy(bomb[1].pos\_x + 7, bomb[1].pos\_y + 4);

            puts("           ");

            bomb[1].pos\_y = 24; // 2번 폭탄의 y값을 초기값으로 변경

            bomb[1].con = FALSE; // 2번 폭탄의 상태를 FALSE로 변경

        }

        break; // switch문 탈출

    }

}

item\_gen.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // item 구조체 사용을 위해 포함

// item 구조체, 생성 시점을 확인하기 위한 int형 변수를 매개변수로 하는 함수

// 아이템 생성을 담당한다.

void item\_gen(Item\* item, int frame\_count)

{

// 아이템 타입 추가시 코드를 좀 단순하게 만들기위해 사용

    int isCreated = FALSE; // 아이템 생성 유무를 확인하기 위한 변수

// 아이템 개수만큼 for문을 반복한다.

    for (int i = 0; i < ITEM\_SIZE; i++)

    {

// 프레임 카운트가 700일 때 아이템을 드랍한다.

        if (frame\_count == 700)

        {

// 무기 파워업 타입의 아이템을 드랍한다.

            item[i].con = TRUE; // 아이템 상태를 TRUE로 변경

            item[i].pos\_x = 10; // x값 저장

            item[i].pos\_y = 0; // y값 저장

            item[i].type = 0; // 타입 0은 무기 파워업

            isCreated = TRUE; // 아이템 생성 유무 변수에 TRUE를 대입

        }

        if (isCreated == TRUE) // 아이템이 생성됬을떄

        {

            gotoxy(item[i].pos\_x, item[i].pos\_y); // 아이템의 xy위치에서

            switch (item[i].type) // 저장된 타입값을 기준으로 switch문에서

            {

            case 0:

                printf("[P]"); // 무기 파워업 타입의 아이템 아이콘을 그려준다.

                break; // switch문 탈출

            }

            break; // for문 탈출

        }

    }

}

item\_move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // item 구조체 사용을 위해 포함

// item 구조체를 매개변수로 하는 함수

// 아이템 움직임을 담당한다.

void item\_move(Item\* item)

{

// 아이템 사이즈 만큼 for문 반복

    for (int i = 0; i < ITEM\_SIZE; i++)

    {

// 아이템 생성 상태 유무를 확인한다.

        if (item[i].con == TRUE)

        {

// 15턴마다 아이템이 떨어진다.

            if (item[i].speed == 15)

            {

                item[i].speed = 0; // 아이템 스피드를 0으로 초기화

// 아이템의 기존 위치를 공백으로 지워준다.

                gotoxy(item[i].pos\_x, item[i].pos\_y);

                puts("   ");

// 아이템 Y값을 증가 시킨다.

                item[i].pos\_y++;

                gotoxy(item[i].pos\_x, item[i].pos\_y);

                switch (item[i].type) // 아이템 타입을 기준으로 하는 switch문

                {

                case 0: // 0타입 아이템을 그려준다.

                    printf("[P]");

                    break; // switch문 탈출

                }

            }

            else // 15번째 턴이 아니라면 턴을 추가해준다.

                item[i].speed++;

        }

    }

}

item\_status.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // 구조체들을 사용하기 위해 포함

// item, player, bullet 구조체와 동작 구별용 int형 변수를 매개변수로 하는 함수

// 아이템 상태를 담당한다.

void item\_status(Item\* item, Player player, Bullet\* bullet, int num)

{

    switch (num) // num을 기준으로하는 switch문

    {

    case 0: // 아이템과 유저가 충돌했을 때

        for (int i = 0; i < ITEM\_SIZE; i++)

        {

// 아이템이 생성된 상태일 때

            if (item[i].con == TRUE)

            {

// 아이템의 Y값과 플레이어의 Y값이 같을 때

                if (item[i].pos\_y == player.pos\_y)

                {

// 플레이어 비행기의 길이와 아이템의 길이가 다르기 때문에 아이템이 플레이어 비행기안에 들어오는지 검사한다.

                    if (item[i].pos\_x >= player.pos\_x && item[i].pos\_x <= player.pos\_x + 5)

                    {

// 아이템의 기존위치를 공백으로 지워준다.

                        gotoxy(item[i].pos\_x, item[i].pos\_y);

                        puts("   ");

                        item[i].con = FALSE; // 아이템의 상태를 FALSE로 바꿔준다.

                        switch (item[i].type) // 아이템 타입을 기준으로 switch문 사용

                        {

                        // 타입을 기준으로 총알들의 타입을 바꿔준다.

                        case 0:

                            bullet[0].type = 1;

                            bullet[1].type = 1;

                            bullet[2].type = 1;

                            bullet[3].type = 1;

                            break; // switch문 탈출

                        case 1:

                            bullet[0].type = 2;

                            bullet[1].type = 2;

                            bullet[2].type = 2;

                            bullet[3].type = 2;

                            break; // switch문 탈출

                        }

                    }

                }

            }

        }

        break; // (num을 기준으로 하는) switch문 탈출

// 아이템이 바닥에 도달했을 때

    case 1:

// 아이템 크기 만큼 for문을 돌려준다.

        for (int i = 0; i < ITEM\_SIZE; i++)

        {

// 아이템이 생성된 상태일 때

            if (item[i].con == TRUE)

            {

// 아이템 y 값이 30 (바닥)일 때

                if (item[i].pos\_y == 30)

                {

// 아이템을 공백으로 지워주고 상태를 FALSE로 바꿔준다.

                    gotoxy(item[i].pos\_x, item[i].pos\_y);

                    puts("   ");

                    item[i].con = FALSE;

                }

            }

        }

        break; // (num을 기준으로 하는) switch문 탈출

    }

}

enemy\_gen.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // enemy 구조체를 사용하기 위해 포함

// enemy 구조체와 게임 진행 시간을 int형 변수를 매개변수로 하는 함수

// 적군의 생성을 담당한다.

void enemy\_gen(Enemy\* enemy, int frame\_cnt)

{

    // item\_gen.c와 같이 다양한 타입의 적을 생성해야 해서

    // 반복적인 코드를 줄이기 위해 bool타입을 따라하는 변수 생성

    int isCreated = FALSE;

    // 적 개수 만큼 for문 반복

    for (int i = 0; i < ENEMY\_SIZE; i++)

    {

    // 진행 시간이 45 ~ 315초 동안 90으로 나눴을 때 나머지가 0일 때

    // i번째 적이 FALSE라면 생성한다.

        if ((enemy[i].con == FALSE && (frame\_cnt >= 45 && frame\_cnt <= 315) && (frame\_cnt - 45) % 90 == 0))

        {

    // 적 비행기의 기본 정보를 입력해준다.

            enemy[i].pos\_x = 6; // 시작 위치

            enemy[i].pos\_y = 0; // 시작 위치

            enemy[i].type = 0; // 적 비행기 모양

            enemy[i].move\_pattern = 0; // 오른쪽 대각선으로 이동 – 일직선 공격

            enemy[i].move\_count = 0; // 공격 빈도를 정하기 위해 사용

            enemy[i].speed = 0; // 움직이는 속도를 조절하기 위해 사용

            enemy[i].health = 5; // 적의 체력

            enemy[i].con = TRUE; // 생성 상태로 변경

            isCreated = TRUE; // 출력을 위해 bool 타입을 따라하는 변수를 TRUE로 변경

        }

    // 진행 시간이 90 ~ 360초 동안 90으로 나눴을 때 나머지가 0일 때

    // i번째 적이 FALSE라면 생성한다.

    // 시간이 다른것과 나머지가 0일 때 생성하는 이유는 순차적으로 생성하여

    // 서로 겹치지 않게 하기 위함이다.

        if ((enemy[i].con == FALSE && (frame\_cnt >= 90 && frame\_cnt <= 360) && (frame\_cnt - 90) % 90 == 0))

        {

            enemy[i].pos\_x = 50; // 시작 위치

            enemy[i].pos\_y = 1; // 시작 위치

            enemy[i].type = 0; // 적 비행기 모양

            enemy[i].move\_pattern = 1; // 왼쪽 대각선으로 이동 – 일직선 공격

            enemy[i].move\_count = 0; // 공격 빈도를 정하기 위해 사용

            enemy[i].speed = 0; // 움직이는 속도를 조절하기 위해 사용

            enemy[i].health = 5; // 적 기체의 체력

            enemy[i].con = TRUE; // 생성 상태로 변경

            isCreated = TRUE; // 출력을 위해 TRUE로 변경

        }

        // 진행 시간이 500일 때 3개가 뭉쳐서 다니는 적 생성

        if (enemy[i].con == FALSE && frame\_cnt == 500)

        {

            // 처음에는 배열처럼 한줄로 저장했으나

            // 버그 수정을 하면서 알아보기 힘들어서 한줄 한줄 입력하는 코드로 변경

            enemy[i].pos\_x = 18; // 시작 위치

            enemy[i].pos\_y = 1; // 시작 위치

            enemy[i].type = 1; // 적 비행기 모양

            enemy[i].move\_pattern = 2; // 아래로 직진 – 오른쪽 대각선으로 공격

            enemy[i].move\_count = 0; // 공격 빈도를 정하기 위해 사용

            enemy[i].speed = 0; // 움직이는 속도를 조절하기 위해 사용

            enemy[i].health = 5; // 적 비행기 체력

            enemy[i].con = TRUE; // i 번째 적을 생성했다는 의미

            // 3기가 뭉쳐서 다녀야하기 때문에 i + 1 번째에도 동일 정보 입력

            enemy[i + 1].pos\_x = 13;

            enemy[i + 1].pos\_y = 0;

            enemy[i + 1].type = 1;

            enemy[i + 1].move\_pattern = 2;

            enemy[i + 1].move\_count = 0;

            enemy[i + 1].speed = 0;

            enemy[i + 1].health = 5;

            enemy[i + 1].con = TRUE;

            // 3기가 뭉쳐서 다녀야하기 때문에 i + 2 번째에도 동일 정보 입력

            enemy[i + 2].pos\_x = 23;

            enemy[i + 2].pos\_y = 0;

            enemy[i + 2].type = 1;

            enemy[i + 2].move\_pattern = 2;

            enemy[i + 2].move\_count = 0;

            enemy[i + 2].speed = 0;

            enemy[i + 2].health = 5;

            enemy[i + 2].con = TRUE;

            isCreated = TRUE; // 출력을 위해 사용

        }

        // 진행 시간이 600일 때 3개가 뭉쳐서 다니는 적 생성

        if (enemy[i].con == FALSE && frame\_cnt == 650)

        {

            enemy[i].pos\_x = 36; // 시작 위치

            enemy[i].pos\_y = 1; // 시작 위치

            enemy[i].type = 1; // 적 비행기 모양

            enemy[i].move\_pattern = 3; // 아래로 직진 – 왼쪽 대각선으로 공격

            enemy[i].move\_count = 0; // 공격 빈도를 정하기 위해 사용

            enemy[i].speed = 0; // 움직이는 속도를 조저랗기 위해 사용

            enemy[i].health = 5; // 적 비행기 체력

            enemy[i].con = TRUE; // i 번째 적을 생성

            // 3기가 뭉쳐서 다녀야하기 때문에 i + 2 번째에도 동일 정보 입력

            enemy[i + 1].pos\_x = 31;

            enemy[i + 1].pos\_y = 0;

            enemy[i + 1].type = 1;

            enemy[i + 1].move\_pattern = 3;

            enemy[i + 1].move\_count = 0;

            enemy[i + 1].speed = 0;

            enemy[i + 1].health = 5;

            enemy[i + 1].con = TRUE;

            // 3기가 뭉쳐서 다녀야하기 때문에 i + 2 번째에도 동일 정보 입력

            enemy[i + 2].pos\_x = 41;

            enemy[i + 2].pos\_y = 0;

            enemy[i + 2].type = 1;

            enemy[i + 2].move\_pattern = 3;

            enemy[i + 2].move\_count = 0;

            enemy[i + 2].speed = 0;

            enemy[i + 2].health = 5;

            enemy[i + 2].con = TRUE;

            isCreated = TRUE; // 출력을 위해 사용

        }

        // 적 기체가 생성되었다면

        if (isCreated == TRUE)

        {

      // i 번째 저장된 x,y 값으로 이동하여

            gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

            switch (enemy[i].type) // type을 기준으로 하는 switch 문에서

            {

            case 0: // 타입이 0이라면 아래 그림을 출력

                printf("<WvW>");

                break; // switch문 탈출

            case 1: // 타입이 1이라면 아래 그림을 출력

                printf("<XVX>");

                break; // switch문 탈출

            }

            break; // for문 탈출

// 하나만 그리고 탈출해도 문제가 없는게

// 뭉쳐서 다니는 기체의 경우 삼각편대 모양으로 출력하기 때문에

// 처음 기체만 화면에 먼저 그려줘도 상관없음

        }

    }

}

enemy\_move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // enemy 구조체를 사용하기 위해 포함

// enemy 구조체를 매개변수로 하는 함수

// 적 비행기의 움직임을 담당한다.

void enemy\_move(Enemy\* enemy)

{

    // 적 사이즈 만큼 for문을 돌린다.

    for (int i = 0; i < ENEMY\_SIZE; i++)

    {

        // 적이 생성된 상태라면

        if (enemy[i].con == TRUE)

        {

            // 기존 위치를 공백으로 지워준다.

            gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

            puts("     ");

            // 패턴이 0 이라면 오른쪽 대각선으로 이동

            if (enemy[i].move\_pattern == 0)

            {

                // 6번 반복할때마다 1번 움직임

                // 자기 턴이 오면 진행

                if (enemy[i].speed == 6)

                {

                    // 움직임 패턴 0은 오른쪽 아래로 내려가는 패턴

                    // 오른쪽으로 2번 아래로 한번으로 움직인다.

                    // 아래로 내려갈 차례인지 옆으로 갈 차례인지 검사

                    if (enemy[i].move\_count == 3)

                    {

                        enemy[i].pos\_y++; // 3번째 움직인은 아래 방향으로 이동이라 Y값만 증가

                        enemy[i].move\_count = 0; // 움직임 횟수를 0으로 초기화

                    }

                    // 1, 2번째 움직인 순서이거나 오른쪽 벽에 붙었을때는 아래쪽으로만 움직인다.

                    else

                    {

                        // 벽쪽으로 갔는지 검사, 벽쪽으로 갔으면 move\_count 올릴 필요 없이 Y값만 증가

                        if (enemy[i].pos\_x > 48)

                            enemy[i].pos\_y++; // 벽에 붙었으니 Y값만 증가

// 1,2 번째 순서라면 오른쪽으로 이동한다.

                        else

                        {

                            enemy[i].pos\_x++; // x값을 증가시켜 오른쪽 이동

                            enemy[i].move\_count++; // 이동 횟수 증가

                        }

                    }

                    // 턴 초기화

                    enemy[i].speed = 0;

                }

// 움직일 턴이 아니라면 speed 값 추가

                else

                {

                    enemy[i].speed++;

                }

            }

            // 패턴 1은 왼쪽 대각선으로 이동

            if (enemy[i].move\_pattern == 1)

            {

                // 자기 턴이 오면 진행

                // 5~15까지 돌려봤을 때 9가 가장 안 겹치고 자연스럽게 지나가서 9로 결정함

                if (enemy[i].speed == 9)

                {

                    // 아래로 내려갈 차례인지 옆으로 갈 차례인지 검사

                    // 패턴 1은 패턴 0과는 다르게 왼쪽으로 한번 아래로 한번으로 움직인다.

                    // 둘다 3번째에 아래로 내려갔을 때 겹치는 문제가 있었음

                    if (enemy[i].move\_count == 2)

                    {

                        enemy[i].pos\_y++; // Y값 증가

                        enemy[i].move\_count = 0; // 움직임 횟수 초기화

                    }

                    else

                    {

                        // 벽쪽으로 갔는지 검사, 벽쪽으로 갔으면 move\_count를 올릴 필요 없이 Y값만 증가한다.

                        if (enemy[i].pos\_x < 10)

                            enemy[i].pos\_y++; // Y값 증가

                        else // 아직 벽쪽이 아니라면 왼쪽으로 이동시킨다.

                        {

                            enemy[i].pos\_x -= 2; // 패턴 0과 이동 횟수가 다르기 때문에 어느정도 속도를 맞추기 위해 패턴1은 한번에 두칸씩 이동

                            enemy[i].move\_count++;

                        }

                    }

                    // 턴 초기화

                    enemy[i].speed = 0;

                }

                else

                {

                    enemy[i].speed++; // 자기 턴이 아니라며 speed 값 추가

                }

            }

            // 패턴 2와 3은 편대 패턴 + 아래로 직진

            if (enemy[i].move\_pattern == 2 || enemy[i].move\_pattern == 3)

            {

                // 패턴 2, 3은 천천히 내려다가 일정 구간부터는 빨리 내려감

                // y값이 12보다 크면 빨리 내려감

                if (enemy[i].pos\_y > 12)

                {

                    // 자기 턴이 오면 진행

                    if (enemy[i].speed == 2)

                    {

                        enemy[i].pos\_y++; // y값 증가

                        // 빨리 내려갈때도 총알을 쏘기 때문에 움직임 횟수를 관리한다, 3번 움직일때마다 총알을 발사한다.

                        if (enemy[i].move\_count == 3) // 3번 움직였으면

                            enemy[i].move\_count = 0; // 0으로 초기화

                        else

                            enemy[i].move\_count++; // 3번째가 아니라면 횟수 추가

                        // 턴 초기화

                        enemy[i].speed = 0;

                    }

                    else

                    {

                        enemy[i].speed++; // 턴이 아니라면 speed 증가

                    }

                }

                Else // y값이 12 미만이면 천천히 움직인다.

                {

                    // 자기 턴이 오면 진행

                    if (enemy[i].speed == 20)

                    {

                        enemy[i].pos\_y++; // Y값 증가

// 총알 발사를 위한 move\_count

                        if (enemy[i].move\_count == 3) // 3번째 움직임이면

                            enemy[i].move\_count = 0; // 초기화

                        else

                            enemy[i].move\_count++; // 아직 3번째가 아니라면 증가

                        // 턴 초기화

                        enemy[i].speed = 0;

                    }

                    else

                    {

                        enemy[i].speed++; // 자기 턴이 아니라면 speed 추가

                    }

                }

            }

// i번째 x,y 값으로 이동한뒤

            gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

            switch (enemy[i].type) // 타입을 기준으로 하는 switch 문에서

            {

            case 0: // 0 타입 그림 출력

                printf("<WvW>");

                break;

            case 1: // 0 타입 그림 출력

                printf("<XVX>");

                break;

            }

        }

    }

}

enm\_bull\_move.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // enemy 구조체를 사용하기 위해 포함

// enemy 구조체를 매개변수로 하는 함수

// 적 비행기 총알의 움직임을 담당한다.

void enm\_bull\_move(Enemy\* enemy)

{

    // 적 개수 만큼 for문 반복

    for (int i = 0; i < ENEMY\_SIZE; i++)

    {

        // 적 총알 개수 만큼 for문 반복

        for (int j = 0; j < ENEMY\_BUL\_SIZE; j++)

        {

            // i번째 적의 총알 j 가 생성된 상태이고 적의 패턴이 0이나 1이라면

            if (enemy[i].bul\_con[j] == TRUE && (enemy[i].move\_pattern == 1 || enemy[i].move\_pattern == 0))

            {

// 총알의 기존 위치를 공백으로 지워준다.

                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                puts("  ");

// 총알의 Y값을 증가시킨다.

                enemy[i].bul\_pos\_y[j]++;

// 변경된 위치에 총알을 그려준다.

                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                printf("\*");

            }

            // i번째 적의 총알 j 가 생성된 상태이고 적의 패턴이 2라면

            if (enemy[i].bul\_con[j] == TRUE && enemy[i].move\_pattern == 2)

            {

// 총알의 기존 위치를 공백으로 지워준다.

                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                puts("  ");

// 패턴 2의 총알은 왼쪽에서 오른쪽 대각선 공격이기 때문에

// x값과 y값을 각각 증가시켜준다.

                enemy[i].bul\_pos\_x[j]++;

                enemy[i].bul\_pos\_y[j]++;

// 변경된 위치에 총알을 그려준다.

                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                printf("¤");

            }

            // i번째 적의 총알 j 가 생성된 상태이고 적의 패턴이 3라면

            if (enemy[i].bul\_con[j] == TRUE && enemy[i].move\_pattern == 3)

            {

// 총알의 기존 위치를 공백으로 지워준다.

                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                puts("  ");

// 패턴 3의 총알은 오른쪽에서 왼쪽 대각선 공격이기 때문에

// x값은 차감 y값은 증가시켜준다.

                enemy[i].bul\_pos\_x[j]--;

                enemy[i].bul\_pos\_y[j]++;

// 변경된 위치에 총알을 그려준다.

                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                printf("¤");

            }

        }

    }

}

enm\_bull\_status.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // 이미 정의한 구조체를 사용하기 위해 포함

// player 구조체를 리턴하고 enemy, player, bomb, bullet 구조체와 동작 확인용 int 변수를 매개변수로 하는 함수

// 적 비행기 총알의 상태 관련 처리들을 담당한다.

Player enm\_bull\_status(Enemy\* enemy, Player player, Bomb\* bomb, Bullet\* bullet, int num)

{

    switch (num) // num을 기준으로 switch문에서 어떤 동작인지 구별

    {

    // 적 총알 발사

    case 0:

        // 적 개수만큼 for문 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

            // i번째 적이 생성 상태라면

            if (enemy[i].con == TRUE)

            {

                // 적 총알 개수만큼 for문을 반복한다.

                for (int j = 0; j < ENEMY\_BUL\_SIZE; j++)

                {

                    // 적의 패턴이 0이나 1이라면

                    if (enemy[i].move\_pattern == 1 || enemy[i].move\_pattern == 0)

                    {

                        // 총알 속도가 없으면 움직일때마다 총알을 발사해서 게임이 불가능함

                        // 총알 속도가 60 이상이고 움직임 횟수가 1이나 3일때

                        if (enemy[i].bul\_speed > 60 && (enemy[i].move\_count == 1 || enemy[i].move\_count == 3))

                        {

                            // 적 총알의 X 값 설정

                            enemy[i].bul\_pos\_x[j] = enemy[i].pos\_x + 2;

                            // 적 총알의 Y 값 설정

                            enemy[i].bul\_pos\_y[j] = enemy[i].pos\_y + 1;

                            enemy[i].bul\_con[j] = TRUE; // 총알 상태 변경

                            enemy[i].bul\_speed = 0; // 총알 속도 초기화

                            // 총알 위치에 가서 그려준다.

                            gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                            puts("\*");

                        }

                    }

                    // 적 패턴이 2나 3일떄

                    else if (enemy[i].move\_pattern == 2 || enemy[i].move\_pattern == 3)

                    {

                        // 총알 속도가 60 이상이고 움직임 횟수가 3일때

                        if (enemy[i].bul\_speed > 60 && enemy[i].move\_count == 3)

                        {

                            // 적 총알의 X, Y 값 설정

                            enemy[i].bul\_pos\_x[j] = enemy[i].pos\_x + 2;

                            enemy[i].bul\_pos\_y[j] = enemy[i].pos\_y + 1;

                            enemy[i].bul\_con[j] = TRUE; // 상태 변경

                            enemy[i].bul\_speed = 0; // 총알 속도 초기화

                            // 총알 위치에 가서 그려준다.

                            gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                            puts("¤");

                        }

                    }

                }

                enemy[i].bul\_speed++; // 총알 속도를 올려준다.

            }

        }

        break; // switch문 탈출

    // 적 총알 플레이어 충돌

    case 1:

// 적 개수 만큼 for문 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

// 적 총알 개수 만큼 반복

            for (int j = 0; j < ENEMY\_BUL\_SIZE; j++)

            {

// 적 총알과 플레이어의 Y값이 동일할 때

                if (enemy[i].bul\_con[j] == TRUE && enemy[i].bul\_pos\_y[j] == player.pos\_y)

                {

// 적 총알이 플레이어 비행기 크기 안으로 들어올 때

                    if ((enemy[i].bul\_pos\_x[j] >= player.pos\_x) && (enemy[i].bul\_pos\_x[j] <= player.pos\_x + 5))

                    {

// 적 총알을 지워준다.

                        gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                        puts("  ");

                        enemy[i].bul\_con[j] = FALSE; // 총알 상태를 FALSE로 변경

                        if (player.health > 0) // 풀레이어의 체력이 남았다면

                            player.health--; // 체력 차감

                        else // 체력이 남지 않았을 때

                        {

                            if (player.life > 0) // 생명이 남아있다면

                            {

                                player.life--; // 생명 차감

                                player.health = 3; // 1 생명 = 3 체력

// 적 총알을 지운다.

                                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                                puts("     ");

// 플레이어를 처음 시작 위치로 이동

                                player.pos\_x = 26;

                                player.pos\_y = 26;

// 원작에서도 한번 터지면 폭탄 기능을 활성화함

                                bomb[0].con = TRUE; // 폭탄 활성화

                                bomb[1].con = TRUE; // 폭탄 활성화

                            }

                        }

                    }

                }

            }

        }

        break; // switch문 탈출

    // 적 총알 바닥 도달

    case 2:

// 적 개수만큼 for문 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

// 적 총알 개수 만큼 for문 반복

            for (int j = 0; j < ENEMY\_BUL\_SIZE; j++)

            {

// 적 총알이 생성된 상태이고 y값이 29보다 클 때

                if (enemy[i].bul\_con[j] == TRUE && enemy[i].bul\_pos\_y[j] > 29)

                {

// 적 총알을 지워준다.

                    gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                    puts("  ");

                    enemy[i].bul\_con[j] = FALSE; // 적 총알의 상태를 FALSE로 변경

                }

            }

        }

        break; // switch문 탈출

    // 적 총알 플레이어 총알 충돌

    case 3:

// 적 개수 만큼 for문 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

// 적 총알 개수 만큼 for문 반복

            for (int j = 0; j < ENEMY\_BUL\_SIZE; j++)

            {

// 플레이어 총알 개수 만큼 for문 반복

                for (int k = 0; k < BULLET\_SIZE; k++)

                {

// 적 총알과 플레이어 총알 둘다 TRUE 상태일때

                    if (enemy[i].bul\_con[j] == TRUE && bullet[k].con == TRUE)

                    {

// 적 총알과 플레이어 총알의 y값이 동일할 때

                        if (enemy[i].bul\_pos\_y[j] == bullet[k].pos\_y)

                        {

// 적 총알과 플레이어 총알의 x 값이 동일할 때

                            if (enemy[i].bul\_pos\_x[j] == bullet[k].pos\_x)

                            {

// 총알 위치가 겹치니 총알이 있던 자리를 공백으로 지움

                                gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                                puts("  ");

                                enemy[i].bul\_con[j] = FALSE; // 적 총알 FALSE

                                bullet[k].con = FALSE; // 플레이어 총알 FALSE

                            }

                        }

                    }

                }

            }

        }

        break; // switch문 탈출

    // 폭탄 발사시 적 총알 폭탄 앞에서 삭제

    case 4:

// 폭탄 1이나 2가 TRUE 상태일때

        if (bomb[0].con == TRUE || bomb[1].con == TRUE)

        {

// 적 개수 만큼 반복

            for (int i = 0; i < ENEMY\_SIZE; i++)

            {

// 적 총알 개수만큼 반복

                for (int j = 0; j < ENEMY\_BUL\_SIZE; j++)

                {

// 적 총알이 TRUE 상태이고 적 총알의 Y값이 폭탄의 Y값과 같거나 클 때

                    if (enemy[i].bul\_con[j] == TRUE && (enemy[i].bul\_pos\_y[j] >= bomb[0].pos\_y || enemy[i].bul\_pos\_y[j] >= bomb[1].pos\_y))

                    {

// 총알이 있던 자리를 공백으로 지움

                        gotoxy(enemy[i].bul\_pos\_x[j], enemy[i].bul\_pos\_y[j]);

                        puts("  ");

                        enemy[i].bul\_con[j] = FALSE; // 적 총알 FALSE

                    }

                }

            }

        }

        break; // switch문 탈출

    }

    return player; // 적 총알이 플레이어와 충돌했을 때 플레이어의 x,y 값에 변동이 있기 때문에 player를 리턴한다.

}

enm\_status.c

#include <stdio.h> // 표준 라이브러리 함수들을 포함하는 헤더, puts 사용

#include <stdlib.h> // rand, srand를 사용하기 위해 포함

#include <time.h> // time을 사용하기 위해 포함

#include "common.h" // gotoxy를 사용하기 위해 포함

#include "game.h" // 이미 정의한 구조체를 사용하기 위해 포함

// player 구조체를 리턴하고 enemy, player, bomb, bullet 등 구조체와 동작 확인용 int 변수를 매개변수로 하는 함수

// 적 비행기의 상태 관련 기능을 담당합니다.

Player enm\_status(Enemy\* enemy, Bomb\* bomb, Bomb\_blt\* bomb\_bul, Bomb\_blt\* bomb\_bul2, Player player, Bullet\* bullet, int num)

{

    switch (num) // num을 기준으로 switch문에서 동작 구분을 한다.

    {

    // 적군이 바닥에 도달하면 삭제

    case 0:

// 적 개수 만큼 for문 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

// 적이 TRUE 상태이고 y값이 29일 때 (30으로 하면 화면 전체가 밀리는 버그가 있음)

            if (enemy[i].con == TRUE && enemy[i].pos\_y == 29)

            {

// 적이 있던 자리를 공백으로 지워줌

                gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

                puts("     ");

                enemy[i].pos\_x = 0; // x값 초기화

                enemy[i].pos\_y = 0; // y값 초기화

                enemy[i].con = FALSE; // 적 상태를 FALSE로 변경

            }

        }

        break; // switch 문 탈출

    // 플레이어 - 적 충돌

    case 1:

// 적 개수 만큼 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

// 플레이어의 y값과 적의 y값이 같을 때

            if (player.pos\_y == enemy[i].pos\_y)

            {

// 적 비행기가 플레이어 비행기 사이즈 안으로 들어오는지 검사

                if ((enemy[i].pos\_x <= player.pos\_x && enemy[i].pos\_x + 5 >= player.pos\_x) && enemy[i].pos\_x - 4 <= player.pos\_x)

                {

// 플레이어의 생명이 남아있다면

                    if (player.life > 0)

                    {

                        player.life--; // 생명 차감

                        player.pos\_x = 26; // 초기 시작 X 값으로 변경

                        player.pos\_y = 26; // 초기 시작 y값으로 변경

// 플레이어가 있던 자리를 공백으로 지워줌

                        gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

                        puts("      ");

                        enemy[i].pos\_x = 0; // 적의 x 값 초기화

                        enemy[i].pos\_y = 0; // 적의 y 값 초기화

                        enemy[i].con = FALSE; // 적 상태를 FALSE로 변경

                        bomb[0].con = TRUE; // 폭탄 활성화

                        bomb[1].con = TRUE; // 폭탄 활성화

                        break; // for문 탈출

                    }

                    else // 플레이어의 생명이 없다면

                    {

                        // 게임오버

                    }

                }

            }

        }

        break; // switch문 탈출

    // 총알 - 적 충돌

    case 2:

        srand(time(NULL)); // rand의 seed 값을 컴퓨터 시간으로 설정

// 총알 개수 만큼 반복

        for (int i = 0; i < BULLET\_SIZE; i++)

        {

// 적 개수 만큼 반복

            for (int j = 0; j < ENEMY\_SIZE; j++)

            {

// 플레이어의 총알과 적군이 TRUE 상태이고 Y값이 같을 때

                if ((bullet[i].con == TRUE && enemy[j].con == TRUE) && bullet[i].pos\_y == enemy[j].pos\_y)

                {

// 플레이어의 총알이 적 비행기 사이즈 안에 들어갈 때

                    if ((enemy[j].pos\_x <= bullet[i].pos\_x - 1) && (enemy[j].pos\_x + 5) >= bullet[i].pos\_x + 1)

                    {

// 적의 체력이 남아있다면

                        if (enemy[i].health > 0)

                        {

// 총알의 위치로 이동해서

                            gotoxy(bullet[i].pos\_x, bullet[i].pos\_y);

// 램덤값을 기준으로 switch 문에서

                            switch (rand() % 5 + 1)

                            {

// 값에 따라 아래 문자들을 출력한다.

// 적의 데미지 효과를 구현

                            case 1:

                                puts("0");

                                break;

                            case 2:

                                puts("o");

                                break;

                            case 3:

                                puts("☆");

                                break;

                            case 4:

                                puts("★");

                                break;

                            case 5:

                                puts("♧");

                                break;

                            }

                            enemy[i].health--; // 적의 체력을 차감시킨다.

                        }

                        else // 적의 체력이 없다면

                        {

// 적이 있던 자리를 공백으로 지움

                            gotoxy(enemy[j].pos\_x, enemy[j].pos\_y);

                            puts("      ");

// 적의 x,y 값을 초기화하고 상태를 FALSE로 변경

                            enemy[j].pos\_x = 0;

                            enemy[j].pos\_y = 0;

                            enemy[j].con = FALSE;

// 총알이 있던 자리도 공백으로 지우고 FALSE로 변경

                            gotoxy(bullet[i].pos\_x, bullet[i].pos\_y);

                            puts("  ");

                            bullet[i].con = FALSE;

                        }

                    }

                }

            }

        }

        break; // switch문 탈출

    // 폭탄 총알 - 적 충돌

    case 3:

// 적 개수 만큼 반복

        for (int i = 0; i < ENEMY\_SIZE; i++)

        {

// 폭탄 총알 개수 만큼 반복

            for (int j = 0; j < BOMB\_BUL\_SIZE; j++)

            {

// 폭탄1의 총알과 적이 TRUE 상태일때 y값이 같다면

                if ((bomb\_bul[j].con == TRUE && enemy[i].con == TRUE) && bomb\_bul[j].pos\_y == enemy[i].pos\_y)

                {

// 적이 있던 자리를 공백으로 지워줌

                    gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

                    puts("      ");

                    enemy[i].pos\_x = 0; // x 값 초기화

                    enemy[i].pos\_y = 0; // y 값 초기화

                    enemy[i].con = FALSE; // 상태를 FALSE로 변경

                }

// 폭탄2의 총알과 적이 TRUE 상태일때 y값이 같다면

                if ((bomb\_bul2[j].con == TRUE && enemy[i].con == TRUE) && bomb\_bul2[j].pos\_y == enemy[i].pos\_y)

                {

// 적이 있던 자리를 공백으로 지워줌

// 적의 x,y를 초기화하고 FALSE로 변경함

                    gotoxy(enemy[i].pos\_x, enemy[i].pos\_y);

                    puts("      ");

                    enemy[i].pos\_x = 0;

                    enemy[i].pos\_y = 0;

                    enemy[i].con = FALSE;

                }

            }

        }

        break; // switch문 탈출

    }

    return player; // player 구조체는 주소를 넘기는게 아니라 매개변수를 복사해서 넘기기 때문에 return 해주지 않으면 여기서 바뀐 player 정보는 game.c에서 갱신되지 않기 때문에 player를 리턴 해줘야 한다.

}

|  |
| --- |
| 적군 생성, 타입별 움직임, 아이템 생성, 움직임 등등 코드를 한번 정리하거나 개선을 해줘야 다른 타입 추가를 편하게 할 수 있을 것 같다. |