**객체지향프로그래밍**

**중간고사 대체 과제**

**최종보고서**

**이름 : 임재도**

**학번 : 32193797**

**학과 : 소프트웨어학과**

**필요한 함수(수업 중 배운 내용 외에 필요한 것들)**

**>> 관련 프로젝트로는 워낙 많은 것들이 존재합니다. 게임하면 떠오르는 가장 대표적인 것이 고전 슈팅게임이기 때문입니다. 그러한 프로젝트들을 참고하지 않았으며, 그저 그 게임 자체만을 플레이했던 경험을 바탕으로 필요한 것들을 계획서에 적었고, 그들을 배운 내용을 바탕으로 코드로 옮겼습니다. 그러다 보니 프로그램 자체가 최대한 깔끔하게 짠다고 짜보았으나, 불필요하거나 효율적이지 못한 로직으로 구성되어 있을 수 있습니다.**

Void gotoxy(int x, int y) : C/C++은 GUI 방식이 아닌 콘솔창을 기반으로 하기 때문에 출력 역시 콘솔창을 기준으로 처리해야 한다. 처음에는 공백을 이용하여 하나씩 출력할까 하였지만, C++에서 제공하는 함수 중 콘솔창에서 좌표를 바탕으로 값을 다루는 것이 있었고, 이것이 그 함수다. 그렇기에 이를 활용한다. 이는 Windows.h 에 포함되어 있다.

사용에 앞서서 이는 사용자 정의 함수이기 때문에

COORD pos = {x,y};

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

의 정의 선언이 필요하다.

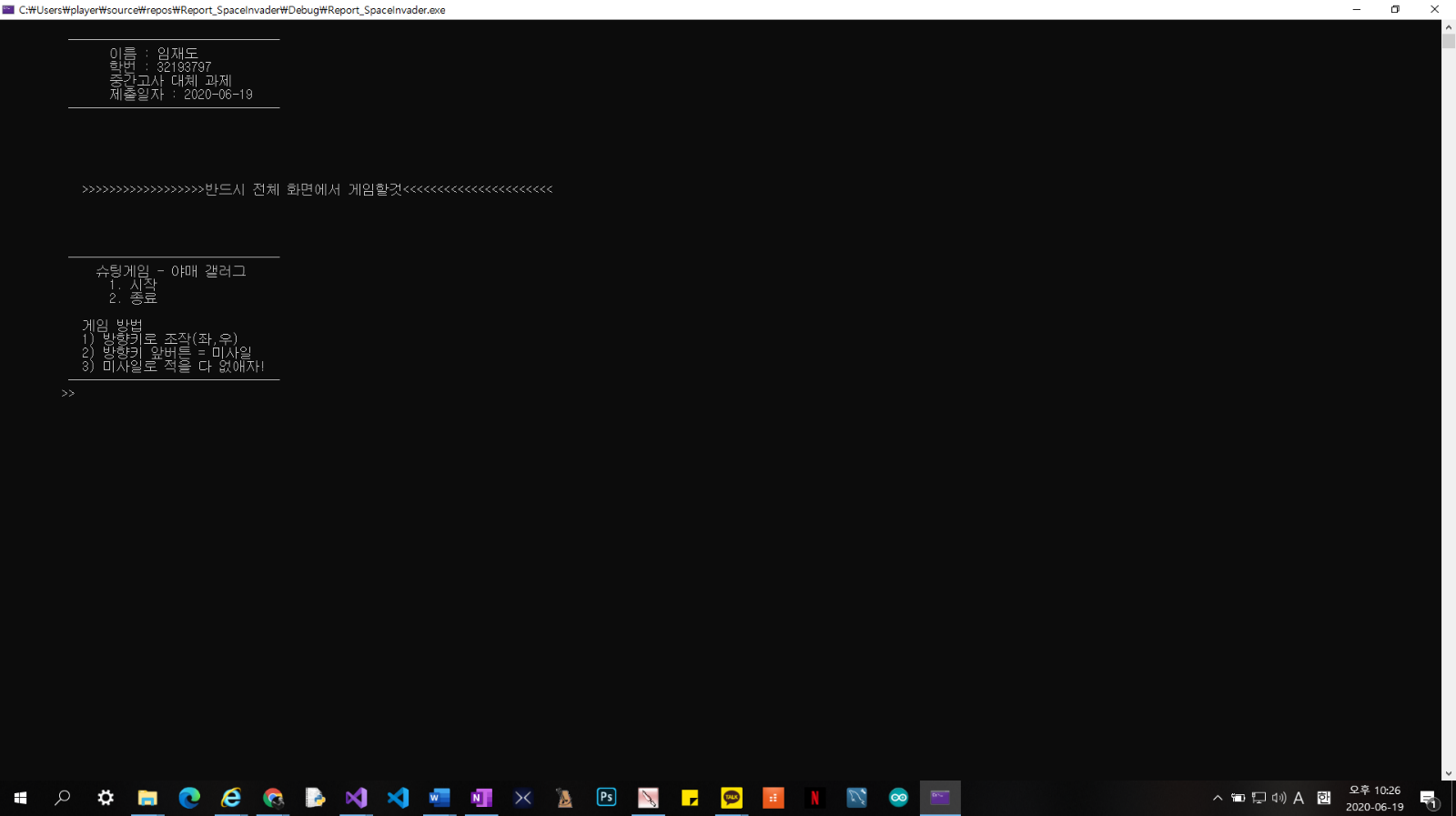
Int kbhit(void) : scanf, cin 등은 값을 입력 받기 전까지 프로그램이 필연적으로 정지할 수 밖에 없다. 갤러그 즉, 스페이스 인베이더 프로그램을 만들기 위해서는 몬스터나 유저가 계속해서 움직여야만 한다. 그렇기에 입력받든 안받든 프로그램이 동작하게 하는 이 함수는 필수적이다. (1은 참, 0은 거짓으로 반환한다.)

Sleep(int) : 말 그대로 잠깐 멈추게 프로그램을 재워주는 함수. int에는 시간이 들어가며, 1000이 1초이다.

System(“cls”) : system() 함수는 콘솔창에 ()안에 있는 명령어를 직접 입력하여 주는 함수이다. 프로그램 동작 시 출력의 원리는 계획서에서 화면을 모두 지우고 거기에 출력함수들을 이용해서 출력하는 형태로 진행할 것이라고 명시한 바가 있다. 이를 위해서 특정 동작이 이루어지면 콘솔을 전부 지워야만 하는 데 이를 실현시키기 위해서 추가한 함수이다.

CONSOLE\_CURSOR\_INFO : 콘솔창에서 커서에 대한 설정을 할 수 있는 클래스이다. 그 맴버로 dwSize, bVisible 이 있으며 전자는 커서크기, 후자는 커서가 보일지 안보일지 여부이다. 게임하는 데 있어서 커서는 보일 필요가 없으므로 bVisible 처리를 해 주었다.

**프로그램에 대한 설명**



인터페이스와 게임 화면이다. 구현을 하기 전에 설계 단계에서 진행했던 기획안과는 다소 차이가 존재한다.

프로그램의 기본적인 원리는 방향키로 플레이어블 캐릭터를 좌, 우로 움직일 수 있게 하였으며, 방향키 앞 버튼을 누르면 미사일이 발사되며, 이것이 0으로 표기되는 몬스터와 만나면 파괴가 된다.

총알은 게임의 난이도 조절을 위해서 최대 2개까지 발사할 수 있게 하였다. 탄환은 비행기가 있는 자리에서 발사가 되며, 괴물에 닿을 시 괴물은 사라지고 1점을 얻게 되며, 괴물에 닿지 못하였을 시 벽 끝에 가야 탄환은 재 장전된다.

**프로그램 설계 코드.**

코드는 2가지 방식으로 구현하였는데, 클래스를 통한 객체 활용과 전역함수를 이용한 설계 방식으로 구현하였다. 이때, 전역 함수를 사용한 이유는 클래스를 활용하여 구현을 하여도 되지만, C++은 C와 연계성을 가지기에 이를 적극 활용해서 각 부분에 있어서 더 편한 구현 방식을 유동적으로 활용해보고자 했기 때문이다.

다만, 이렇게 하다 보니 클래스와 함수 간의 변수를 공유하는 일이 있을 때마다 전역변수를 선언하여 프로그램을 구동하게 되었는데, 그러다 보니 파일 구동에서 약간의 오차가 발생한 경우 그 오차를 정확히 집어 내기 힘든 경우가 생겼으며, 쓸데없는 코드가 늘어나기도 하였다. 최대한 수정을 하였으나, 근본적으로 friend를 사용하여 전역변수를 class내의 변수에 접근하게 하는 식으로 하여 작동시키는 것이 아니라면, 수정할 수 없는 문제임을 깨닫는 일도 발생하였는데, 이는 당초 설계의도와는 어긋나기에 그리고, 지난 과제 들에서 충분히 실습한 내용이기에 이런 문제에도 불구하고, 전역함수와 class를 병행해서 사용하는 방식을 선택하였다.

코드 설계 방식과는 별개로, 코드는 4가지 파일로 구성이 되어있다. 클래스와 전역함수가 어떤 것이 어떻게 작동하는 지 보여주는 틀인 Frame.h 헤더파일에 클래스와 함수의 선언부를 두었고, class.cpp에는 클래스의 정의 부분을, Function.cpp에는 함수의 정의 부분을, 그리고 마지막으로 main.cpp에는 전체적으로 동작하는 부분을 구현해 두었다.

하나씩 살펴보면 다음과 같다.

**이 부분은 Frame.h로, 클래스와 함수의 선언부가 적혀 있는 헤더파일이다.**

#define \_CRT\_SECURE\_NO\_WARNINGS // 혹시 모를 오류 처리 방지

#define Frame\_H

#include <iostream> // 입출력을 위한 전처리

#include <Windows.h> // gotoxy 를 사용하기 위한 전처리

#include <stdlib.h> // 난수 표현 등을 위한 전처리

#include <time.h> // 난수표현에 있어서 seed 값을 설정하기 위한 전처리

#include <conio.h> // kbhit 등을 사용하기 위한 전처리

#include <stdio.h>

using namespace std;

/\*

전체크기 : 가로 50, 세로 25

\*/

/\*

프로그램 작성에 있어서 구분.

시스템 처리 관련된 부분 : 전역 함수를 바탕으로 설정.

내부 객체들의 움직임 등에 관련된 부분 : 클래스를 바탕으로 설정.

\*/

// 전역변수들 (전역 함수들이나 클래스들간의 데이터를 공유하기 위해 별도로 설정.)

//int x = 20, y = 25; // 기준이 되는 x좌표와 y 좌표 설정. (중심부) 기본 위치 설정.

//int bx, by; // 총알의 좌표

//int bx2, by2; //총알의 좌표

//int mx = 1, my = 5, loc[20] = { 0, };//, loc2[15]; // 몬스터들의 위치 + 모두 0으로 초기화

//int max, min; // 괴물들의 위치를 파악하기 위한 변수

//long long count\_t = 0; // 시간을 측정하기 위한 함수.

//int max\_v = 10; // 괴물의 위치를 설정하기 위한 변수

//int min\_v = 10;

//int count\_b, b1, b2; // 총알 여부 확인

//int count\_m; // 괴물의 수

//int score = 0; // 점수

여기까지는 필요한 헤더파일을 포함시키고, 기본적인 공간 셋팅에 대한 주석과, 함수나 클래스에서 사용할 전역 함수들에 대한 주석을 포함하고 있다.

전역 변수의 경우 헤더파일에서 선언하고, 이를 다시 cpp 파일 등에서 #include “Frame.h”와 같은 방식으로 불러올 경우 부르는 cpp마다 전역변수를 선언하기에, 서로 링크하는 과정 속에서 중복 변수 오류가 발생한다. 그에 따라 직접적으로 헤더 파일에 선언하지는 못하고, 헤더파일을 통해서 프로그램의 전체적인 틀을 이해하기 쉽게 하는 것을 목적으로 주석처리해서 남겨두었다.

// 좌표 처리를 하기 위한 함수

void gotoxy(int x, int y);

// 출력을 담당하는 전역함수

// 출력의 원리가 모든 화면을 지웠다가 새로 띄우고 조금이라도 변화 있을시 다시 지우고 하는 것이기에, 이를 고려해서 전역함수로 설정.

void show(void);

// 게임 인터페이스 - 전역변수 설정

int InterFace(void);

이 부분은 전역함수의 선언 부분이다. 주로 좌표를 다루는 부분, 출력하는 부분, 그리고 인터페이스로 게임의 캐릭터 적인 요소를 조작한다기 보다는 시스템적인 요소를 조작할 때 전역함수를 활용하고자 하였다. 이에 대한 세부 내용은 뒤에서 Function.cpp 에 대해 설명할 때 자세히 명시해 두었다.

// 사용자에 대한 클래스

// 현재는 위치에 대한 부분만 구현. 이후 사용자에게 특별한 기능을 넣고 싶을 때 활용.

class Player {

public:

void move(int); // 사용자의 위치 변환 함수.

};

// 총알에 대한 클래스

// 총알의 움직임, 생성, 위치 배치 등을 담당.

class bullet {

public:

void makeB(void); // 총알 생성하는 함수

void move(void); // 총알의 움직임

};

사용자가 조작할 수 있는 Playable 캐릭터에 대한 조작 부분이다. 여기에는 총알에 대한 부분도 포함이 된다. 상당히 애먹었던 부분인데, 이에 대해서는 뒤에 class.cpp에서 자세히 명시해 두었다.

// 괴물에 대한 클래스

class Enemy {

int count;

public:

//void show(void);

Enemy();

void move(void); // 괴물의 이동에 관한 함수 - 시간이 지남에 따라 움직임.

void location(void); // 괴물의 위치

void decide(void); // 괴물의 최고점 최저점 판별.

};

앞의 사용자 캐릭터에 대한 부분 외에도 괴물에 대한 클래스이다. 괴물은 자동으로 생성되서, 자동으로 움직여야 하기에 이를 판별해주기 위해 별도로 고안해야 했으며, 이를 수시로 키보드 조작에 상관없이 출력해주어야 했다. 이에 대한 설명 역시 뒤에 class.cpp를 설명할 때 자세히 명시해 두었다.

**Function.cpp 에 대한 설명이다.**

#include "Frame.h"

// 전역변수 사용을 위한 선언

extern int x, y; // 기준이 되는 x좌표와 y 좌표 설정. (중심부) 기본 위치 설정.

extern int bx, by; // 총알의 좌표

extern int bx2, by2; //총알의 좌표

extern int mx, my, loc[20];//, loc2[20]; // 몬스터들의 위치 + 모두 0으로 초기화

extern int max, min; // 괴물들의 위치를 파악하기 위한 변수

extern long long count\_t; // 시간을 측정하기 위한 함수.

extern int max\_v; // 괴물의 위치를 설정하기 위한 변수

extern int min\_v;

extern int count\_b, b1, b2; // 총알 여부 확인

extern int count\_m; // 괴물의 수

extern int score; // 점수

이미 헤더파일에서 함수에 대한 선언을 해주었기에 이를 참고하기 위해서 전처리로 헤더파일을 포함시켜 두었다. 또한 전역변수들 역시 선언해주었는데, 다른 파일에서도 동시에 사용을 공유해야 하므로 extern으로 모든 전역변수를 설정해 두었다.

전역변수는 시스템에 표시하기 위한 것을 제외하고는 모두 좌표와 관련된 것들이며, 이는 곧 객체를 조작하고, 이런 내용을 출력하는 것은 전역함수이기 때문에 선언된 것들이다.

bx, by, bx2, by2, count\_b, b1, b2 처럼 b가 들어간 것은 모두 총알에 관련된 것이며, m이 들어간 것은 모두 괴물에 관련된 것, 그 외에 이런 기호가 없는 것은 대부분 사용자 캐릭터에 관한 것이다.

다만, count\_t, max\_v, min\_v, score는 시스템적인 요소와 엮여 있는 변수이며, 이들은 주석에 명시를 해 두었다.

void gotoxy(int x, int y) {

COORD pos = { x,y };

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

}

어떻게 보면 갤러그를 구현하는 데 있어서 필수적인 함수이다. 이 방식이 없었다면 for 문과 cout을 활용해서 x, y 좌표를 입력 받은 후 공백을 바탕으로 출력하고 콘솔을 모두 지우는 형식으로 진행하려고 했었다. 실제로 그렇게 구현을 해봤던 결과, 한 줄 씩 출력이 되기에 사용자 비행기가 한번 움직이는 것만 해도 엄청나게 많은 출력 시간이 걸렸고, 괴물의 움직임을 매번 반응시키고 하기 위해서는 또한 엄청나게 긴 코드가 필요하였다. 긴 코드가 전부 의미 있는 코드라고 하기 보다는, 전부 cout 문을 통한 출력 형태였기 때문에 불필요한 작업이었다.

그렇게 고심과 공부 끝에 알게된 함수가 gotoxy 사용자 함수로 (gotoxy 든 뭐든 상관이 없다. 결국 사용자가 내부의 코드를 바탕으로 선언하면 된다. 다만 범용적으로 gotoxy라는 말을 자주 사용하는 것 같아서 이렇게 사용하였다.) 특정 좌표 부분에 콘솔을 미리 옮겨주는 방식을 통해 구현할 수 있었다.

void show(void) {

system("cls");

cout << "\t 게임 플레이 시간 : " << count\_t << endl;

cout << "\t 총알의 수　　　 : " << count\_b << " 개" << endl;

cout << "\t 괴물의 수　　　 : " << count\_m << " 마리" << endl;

cout << "\t 점수 　 　　　 : " << score << " 점" << endl;

// 괴물 출력

for (int i = 0; i < (sizeof(loc) / sizeof(int)); i++) {

if (loc[i] != 0) { gotoxy(loc[i], my); cout << "O"; }

}

// 총알출력

if (b1 == 1) {

gotoxy(bx, by); cout << "↑";

}

if (b2 == 1) {

gotoxy(bx2, by2); cout << "↑";

}

// 사용자의 비행기

gotoxy(x, y - 1); cout << "　△　";

gotoxy(x, y); cout << "◁□▷";

}

또다른 핵심 부분인 출력 함수이다. Gotoxy를 바탕으로 좌표를 설정하고, 그 좌표부터 쭉 이어서 사용자의 비행기를 포함하여 괴물, 총알 등등 모두 출력이 되게 하였다. 동시에 게임을 하는 와중에 시간 등등도 표시가 되게 하였고, 이는 곧 게임 플레이에서의 인터페이스라고 보아도 될 듯하다.

다만 보면 알겠지만 말 그대로 출력만 하는 함수이기에 동작과 관련된 로직은 없고 모두 좌료나 특정 값을 받아와 그 만큼 움직이거나 출력해주는 형태를 가지고 있다.

어떤 값을 이들에게 보내서 출력할지에 대해서는 각각의 클래스에서 구현을 해 두었으며, 이는 class.cpp 즉, 뒤에서 설명할 예정이다.

int InterFace(void) { // 인터페이스 함수

cout << endl;

cout << "\t ──────────────────────────────" << endl;

cout << "\t 이름 : 임재도 " << endl;

cout << "\t 학번 : 32193797 " << endl;

cout << "\t 중간고사 대체 과제 " << endl;

cout << "\t 제출일자 : 2020-06-19 " << endl;

cout << "\t ──────────────────────────────" << endl << endl << endl << endl;

cout << endl << endl << "\t >>>>>>>>>>>>>>>>>>반드시 전체 화면에서 게임할것<<<<<<<<<<<<<<<<<<<<<< " << endl << endl << endl << endl << endl;

cout << "\t ──────────────────────────────" << endl;

cout << "\t 슈팅게임 - 야매 갤러그 " << endl;

cout << "\t 1. 시작 " << endl;

cout << "\t 2. 종료 " << endl;

cout << "\t " << endl;

cout << "\t 게임 방법 " << endl;

cout << "\t 1) 방향키로 조작(좌,우) " << endl;

cout << "\t 2) 방향키 앞버튼 = 미사일 " << endl;

cout << "\t 3) 미사일로 적을 다 없애자! " << endl;

cout << "\t ──────────────────────────────" << endl;

int select = 0; // 명령을 입력받을 변수.

cout << "\t >> ";

cin >> select;

return select;

}

이 부분은 게임을 처음 접속했을 때 보여지는 인터페이스다. 처음 설계했던 방식과 위치는 다르지만 모든 기능을 포함하고 있다. 제작자의 이름과, 과제인 만큼 과제에 대한 주석, 그리고 게임 시작화면을 보여주고 있다.

게임 방법은 별도로 구현을 해도 되었으나, 불필요하다고 여겨서 메인 화면에 같이 포함을 시켰으며, 메인 화면에서는 시작과 종료 두가지만을 입력 받아 실행시킬 수 있게 설계하였다.

**Class.cpp에 대한 설명이다.**

#include "Frame.h"

//전역변수 사용을 위한 선언

int x = 20, y = 25; // 기준이 되는 x좌표와 y 좌표 설정. (중심부) 기본 위치 설정.

int bx, by; // 총알의 좌표

int bx2, by2; //총알의 좌표

int mx = 1, my = 5, loc[20] = { 0, };//, loc2[20]; // 몬스터들의 위치 + 모두 0으로 초기화

int max, min; // 괴물들의 위치를 파악하기 위한 변수

long long count\_t = 0; // 시간을 측정하기 위한 함수.

int max\_v = 10; // 괴물의 위치를 설정하기 위한 변수

int min\_v = 10;

int count\_b, b1, b2; // 총알 여부 확인

int count\_m; // 괴물의 수

int score = 0; // 점수

이 부분은 앞서 설명하였던 Function.cpp와 동일하다.

// 사용자 파트

void Player::move(int a) { // 사용자의 위치 변환.

if (x > 0 && x < 47) {

x += a; // x좌표에만 변화

}

else if (x == 0 && a == 1) x += a;

else if (x == 47 && a == -1) x += a;

}

사용자 파트인데, 플레이어블 캐릭터의 조작에 관한 것은 main함수에서 이루어지도록 설계를 하였기 때문에 여기에는 단순히 값을 입력 받았을 때 전역변수의 수치를 변화시킬 수 있게 하는 정도로 그쳤다. (조작에 대한 부분은 이후 main.cpp에서 설명하였다.)

// 총알파트

void bullet::makeB(void) {

if (count\_b < 2) // 총알 갯수 최대 2개 제한

{

if (b1 == 0) { // 첫번째 총알의 여부에 따라 2번쨰 총알이 발사되거나 하도록 설정

b1 = 1;

bx = x + 2, by = y - 2;

count\_b++;

}

else if (b1 == 1) {

b2 = 1;

bx2 = x + 2, by2 = y - 2;

count\_b++;

}

}

}

void bullet::move(void) {

if (count\_b > 0) // 총알이 존재할 경우

{

// 첫번쨰 총알이 존재할 경우

if (b1 == 1) {

if (count\_t % 5 == 0) by--; // 일정 시간마다 총알은 위로 향함.

if (by == my) { // 괴물의 y좌표와 동일해졌을때

for (int i = 0; i < 60; i++) {

if (loc[i] != 0) {

if (bx == loc[i]) { // 괴물에 닿는다면

count\_b--; // 총알 소거

b1 = 0;

}

}

}

}

int t = my - 2;

if (by == 4) { // 총알이 괴물에 비해 적당히 위로 올라가면

count\_b--; // 총알 소거

b1 = 0; //

}

}

if (b2 == 1) {

if (count\_t % 5 == 0) by2--; // 일정 시간마다 총알은 위로 향함.

if (by2 == my) {

for (int i = 0; i < 60; i++) {

if (loc[i] != 0) {

if (bx2 == loc[i]) { // 괴물에 닿는다면

count\_b--; // 총알 소거

b2 = 0;

}

}

}

}

int t = my - 2;

if (by2 == 4) { // 총알이 괴물에 비해 적당히 위로 올라가면

count\_b--; // 총알 소거

b2 = 0; //

}

}

}

}

많은 애를 먹인 부분이다. 총알을 단순하게 움직이게 하는 것과 총알을 발사 시키는 것은 그다지 어렵지 않았다. 특정 키를 입력 받았을 때 보이게 하는 변수로 설정해둔 b1이나 b2를 1로 바꾸면 보이게 되기 때문이다. 실제로 이에 대해서 구현해 놓은 부분이 makeB 인데 코드가 굉장히 짧고 간결하다.

다만, 이 게임의 핵심은 총알로 적을 쏴서 죽이는 것이다. 그에 따라 총알이 괴물과 접촉하면 총알과 괴물 둘 다 사라져야 할 필요가 있으며, 그와 동시에 점수를 올리고, 괴물의 수를 줄여야만 하는 필요성이 있었다. 또한 동시에 괴물을 맞추지 못할 경우 총알을 다시 회수해야 하기에 이것 역시 반영을 해 두어야 했다.

괴물과 총알이 접촉했을 시 총알을 사라지게 하는 부분을 어차피 괴물도 사라져야 하니 괴물과 함께 동작하게 구현을 하려고 하였지만, 결국 이 역시 총알에 관한 부분이니 총알에 관한 기능은 오로지 bullet 클래스에서만 이루어지게 하고, 괴물에 관한 기능은 enemy 클래스에서 구현을 하자는 생각 하에 따로 구현을 하였다.

구동 원리는 각 좌표를 전역변수로 선언해 두었기 때문에 총알의 좌표와 괴물의 좌표를 비교해서 b1, b2 값을 조작하는 형태로 구현을 해 두었다.

// 괴물 파트

// 시간초에 따라 괴물의 위치 이동을 위해 좌표 수정.

// 시간, 괴물에 대한 움직임에 대한 설정은 여기서.

Enemy::Enemy() {

srand(time(NULL));

count\_m = rand() % 20 + 1; // 몬스터의 수

count = (sizeof(loc) / sizeof(int));

}

제일 시간이 많이 걸린 부분이 괴물 파트이며, 특히 move를 설계하는 데 정말 시간이 많이 걸렸다. 기본 원리는 다음과 같다. 괴물들의 위치를 배열로 지정을 해 둔다. 배열의 크기는 20개 이며, 여기에 랜덤하게 몬스터가 생성이 된다. 몬스터의 개수까지도 랜덤으로 생성하고 싶었는데, 이를 위해서 생성자에서 count\_m, 즉 생성될 괴물 횟수를 난수로 선언해주었다.

// 괴물의 양옆 끝자락 판단

void Enemy::decide(void) {

//괴물의 위치를 오름차순으로 정렬.

int temp;

for (int i = 0; i < count; i++) {

for (int j = 0; j < count - 1; j++) {

if (loc[j] > loc[j + 1]) {

temp = loc[j];

loc[j] = loc[j + 1];

loc[j + 1] = temp;

}

}

}

max\_v = loc[19]; // 오른쪽 제일 끝 값

int dc = 0;

while (loc[dc] == 0) { dc++; }

min\_v = loc[dc]; // 왼쪽 제일 끝 값

}

// 괴물에게 처음 위치 할당

void Enemy::location(void) {

for (int i = 0; i < count\_m; i++) {

loc[i] = rand() % 39 + 1;

for (int j = 0; j < i; j++) {

if (loc[i] == loc[j]) {

i--; break;

}

}

}

decide();

}

생성되는 몬스터의 개수까지 랜덤으로 하고 싶다 보니, 배열을 활용하는 것에서 그치지 않고 배열을 오름차순으로 정렬해서 조작하는 방식을 도입하였다. 게임 설계, 기획서에서 괴물은 벽에 닿으면 한 칸 내려간 다음 반대방향으로 움직여야만 했다. 이를 위해서는 괴물의 양 끝이 어딘지를 알아야만 했다. 그러나 배열의 크기는 20이지만 그 안에 담기는 의미있는 데이터의 값은 1~20 사이이므로 무조건 배열의 처음과 끝으로 설정하면 안됐었다. 그래서, 배열의 디폴트 값을 0으로 하고 오름차순으로 정렬 후 제일 마지막은 당연히 제일 큰 값일 테니 제일 끝 위치를 가리키며, 0을 제외하고 제일 작게오는 즉, 0이 아니면서 제일 처음에 등장하는 배열의 위치에 있는 수가 곧 제일 왼쪽의 괴물이니 이들을 각각 max\_v, min\_v 변수로 설정해서 이후 괴물이 죽더라도 이들을 계속해서 파악할 수 있게 설계하였다.

void Enemy::move(void) {

Sleep(3);

count\_t++;

if (my == by) { // 총알 1에 괴물이 맞았을 경우.

for (int i = 0; i < count; i++) {

if (loc[i] != 0) {

if (loc[i] == bx) {

loc[i] = 0;

count\_m--;

score++;

decide();

}

}

}

}

if (my == by2) { // 총알 2에 괴물이 맞았을 경우.

for (int i = 0; i < count; i++) {

if (loc[i] != 0) {

if (loc[i] == bx2) {

loc[i] = 0;

count\_m--;

score++;

decide();

}

}

}

}

if (count\_t % 5 == 0) { // 특정 시간 간격마다 움직일 수 있도록.

if (my % 2 == 1) { // 몬스터의 y좌표값이 홀수일 경우 오른쪽으로 이동

for (int i = 0; i < count; i++) {

if (max\_v == 49) {

my++;

break;

}

if (loc[i] != 0) {

loc[i]++;

}

}

if (max\_v < 49) { max\_v++; min\_v++; }

if (my == by) { // 총알 1에 괴물이 맞았을 경우.

for (int i = 0; i < count; i++) {

if (loc[i] != 0) {

if (loc[i] == (bx)) {

loc[i] = 0;

count\_m--;

score++;

decide();

}

}

}

}

if (my == by2) { // 총알 2에 괴물이 맞았을 경우.

for (int i = 0; i < count; i++) {

if (loc[i] != 0) {

if (loc[i] == (bx2)) {

loc[i] = 0;

count\_m--;

score++;

decide();

}

}

}

}

}

else if (my % 2 == 0) { // y 좌표값이 짝수일 경우 왼쪽으로 이동

for (int i = 0; i < count; i++) {

if (min\_v == 1) {

my++;

break;

}

if (loc[i] != 0) {

loc[i]--;

}

}

if (min\_v > 1) { min\_v--; max\_v--; }

if (my == by) { // 총알 1에 괴물이 맞았을 경우.

for (int i = 0; i < count; i++) {

if (loc[i] != 0) {

if (loc[i] == (bx)) {

loc[i] = 0;

count\_m--;

score++;

decide();

}

}

}

}

if (my == by2) { // 총알 2에 괴물이 맞았을 경우.

for (int i = 0; i < count; i++) {

if (loc[i] != 0) {

if (loc[i] == (bx2)) {

loc[i] = 0;

count\_m--;

score++;

decide();

}

}

}

}

}

}

}

괴물의 이동 부분임과 동시에, 총알에 맞을 때 죽는 것까지 포함하고 있다. 코드가 다소 복잡해 보일 수 있으나 결국 구조는

총알에 맞으면 괴물은 죽음 -> 오른쪽으로 움직임 -> 오른쪽으로 움직일 때 총알에 맞으면 괴물은 죽음 -> 왼쪽으로 움직임 -> 왼쪽으로 움직일 때 총알에 맞으면 괴물은 죽음.

이 순서로 짜여져 있는 코드이다.

기본적으로 움직임은 x좌표와 y좌표에 +-를 해주는 형태로 구성되어 있다.

앞서 코드 중 decide() 코드가 제일 왼쪽, 제일 오른쪽에 있는 괴물의 위치를 파악하는 코드이다. 그래서 이들이 벽으로 설정한 좌표에 닿으면 y좌표가 +되도록 하였으며, 그게 아니라면 x좌표가 +- 되도록 설계되었다. 동시에 만약 총알을 맞아 괴물이 죽을 경우, 생존한 괴물의 수가 -가 되고, 점수는 +가 되며, 다시금 제일 왼쪽, 오른쪽에 있는 괴물의 위치를 파악하도록 하였다.

**마지막으로 main.cpp이다.**

#include "Frame.h"

// 전역변수 사용을 위한 선언

extern int x, y; // 기준이 되는 x좌표와 y 좌표 설정. (중심부) 기본 위치 설정.

extern int bx, by; // 총알의 좌표

extern int bx2, by2; //총알의 좌표

extern int mx, my, loc[20];//, loc2[20]; // 몬스터들의 위치 + 모두 0으로 초기화

extern int max, min; // 괴물들의 위치를 파악하기 위한 변수

extern long long count\_t; // 시간을 측정하기 위한 함수.

extern int max\_v; // 괴물의 위치를 설정하기 위한 변수

extern int min\_v;

extern int count\_b, b1, b2; // 총알 여부 확인

extern int count\_m; // 괴물의 수

extern int score; // 점수

이 부분은 앞과 동일하다.

// 메인함수 - 기본적인 기능들 수행

int main(void) {

int select;

select = InterFace(); // 인터페이스 실행

if (select == 1) {

// 게임에 들어가고 부터는 커서가 보여선 안됨

CONSOLE\_CURSOR\_INFO cursor = { 0, }; // 커서를 숨기기 위한 객체 선언

cursor.dwSize = 1; // 커서 사이즈 설정

cursor.bVisible = FALSE; // 커서 보이지 않게 셋팅

SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor); //설정한 내용 반영.

srand(time(NULL)); // 랜덤 난수 설정

Player P;

Enemy E;

bullet B;

E.location(); // 처음에 괴물 생성

show(); // 출력

char moveP; // 방향키를 입력받기 위한 변수

while (true) {

E.move(); // 여기 내부에 delay 등이 존재해서 시간처리도 같이 됨.

B.move();

if (\_kbhit()) {

moveP = \_getch();

switch (moveP) {

case 75: // 왼쪽방향키

P.move(-1);

show();

break;

case 77: // 오른쪽 방향키

P.move(1);

show();

break;

case 72: // 위를 누르면 총알발사

B.makeB();

show();

break;

default:

show();

break;

}

}

show();

if (count\_m == 0) { // 승리조건

system("cls");

cout << endl << "──────────────────────" << endl;

cout << endl << "\t 승리!" << endl;

cout << "\t 게임종료!" << endl;

cout << endl << "──────────────────────" << endl;

break;

}

if (my == (y - 3)) { // 패배조건, 괴물이 사용자에게 다가왔을떄

system("cls");

cout << endl << "──────────────────────" << endl;

cout << endl << "\t 패배!" << endl;

cout << "\t 게임종료!" << endl;

cout << endl << "──────────────────────" << endl;

break;

}

}

}

else if (select == 2) {

cout << endl << endl << "\t > 게임 종료" << endl;

return 0;

}

}

메인 함수는 다음과 같은 기능을 수행한다.

1. 프로그램의 동작

2. 인터페이스 출력

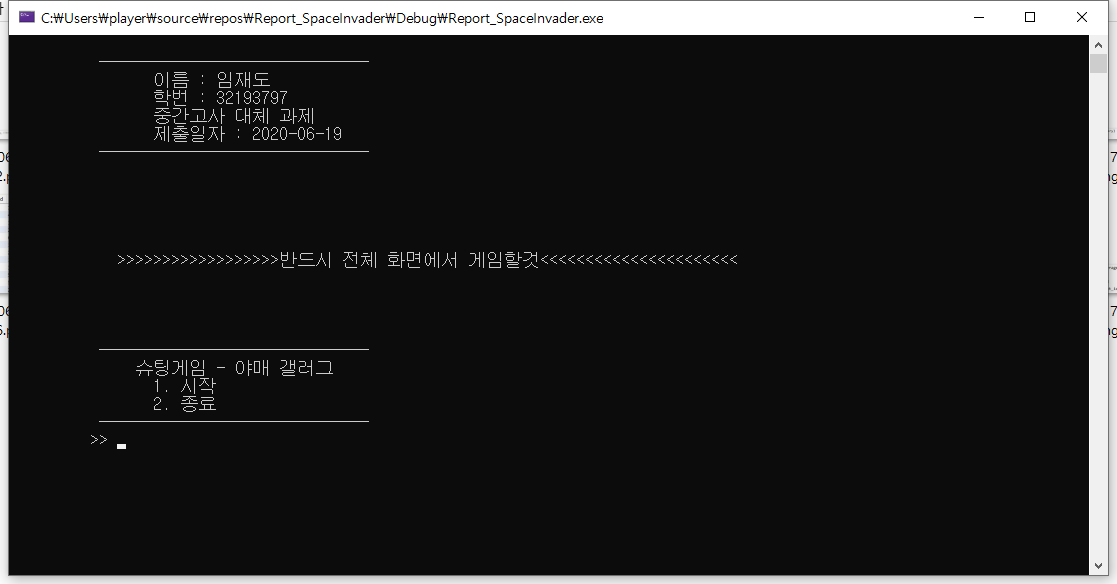
3. 게임 출력 및 방향키로 조작을 할 수 있게 함.

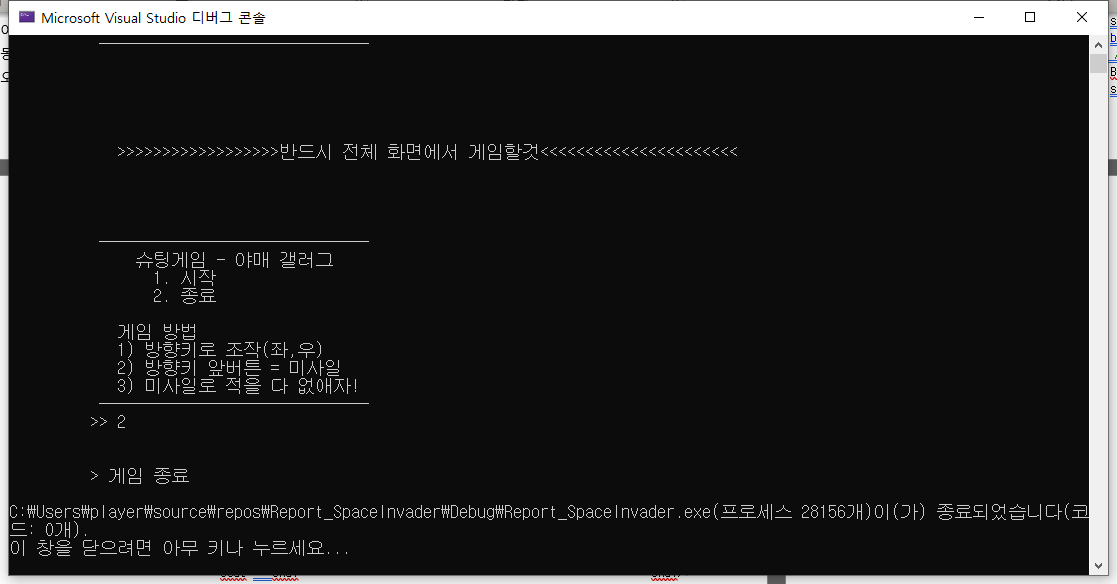
4. 총알을 발사하였을 때 그에 대한 반응들을 출력.

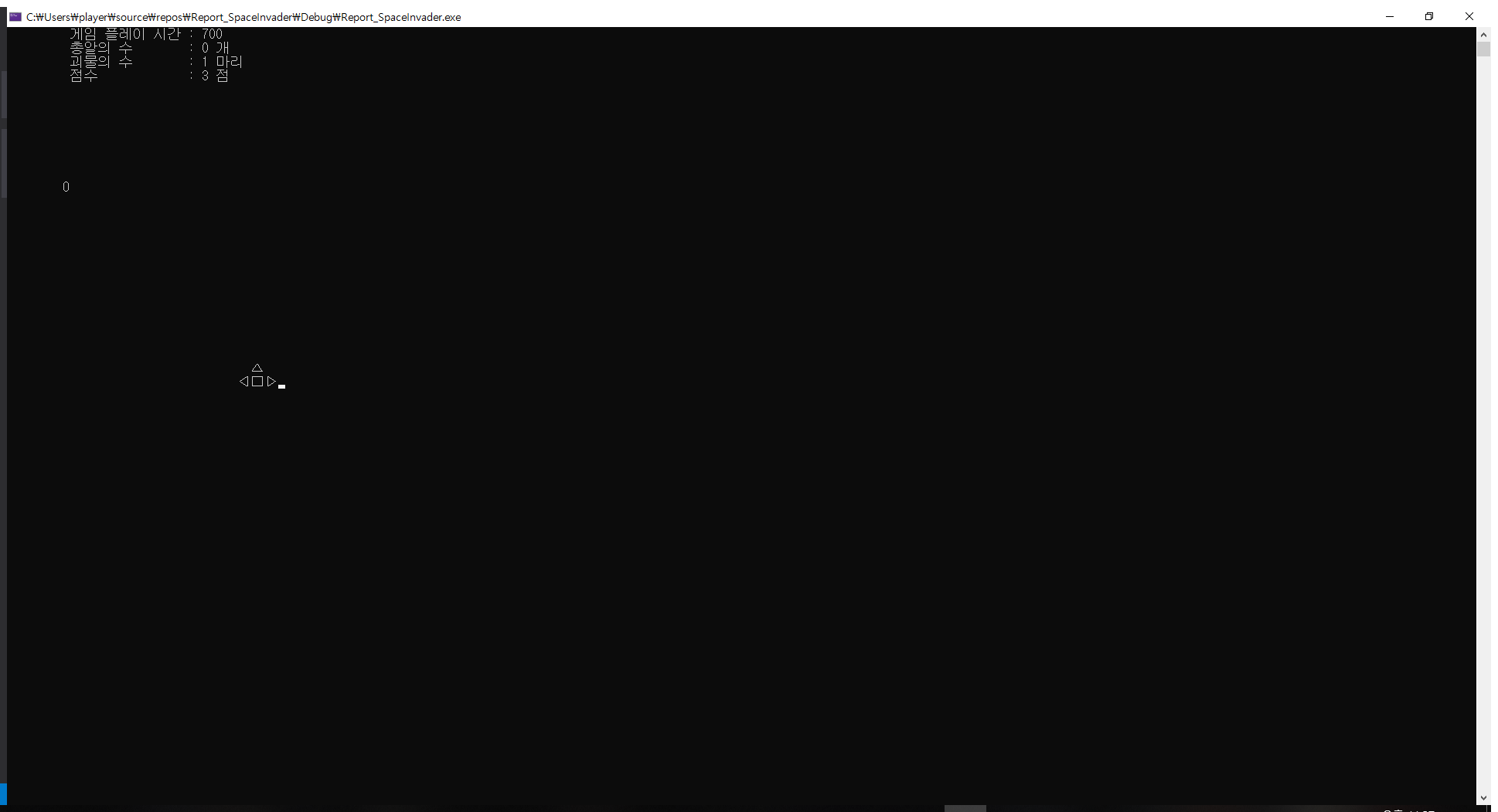
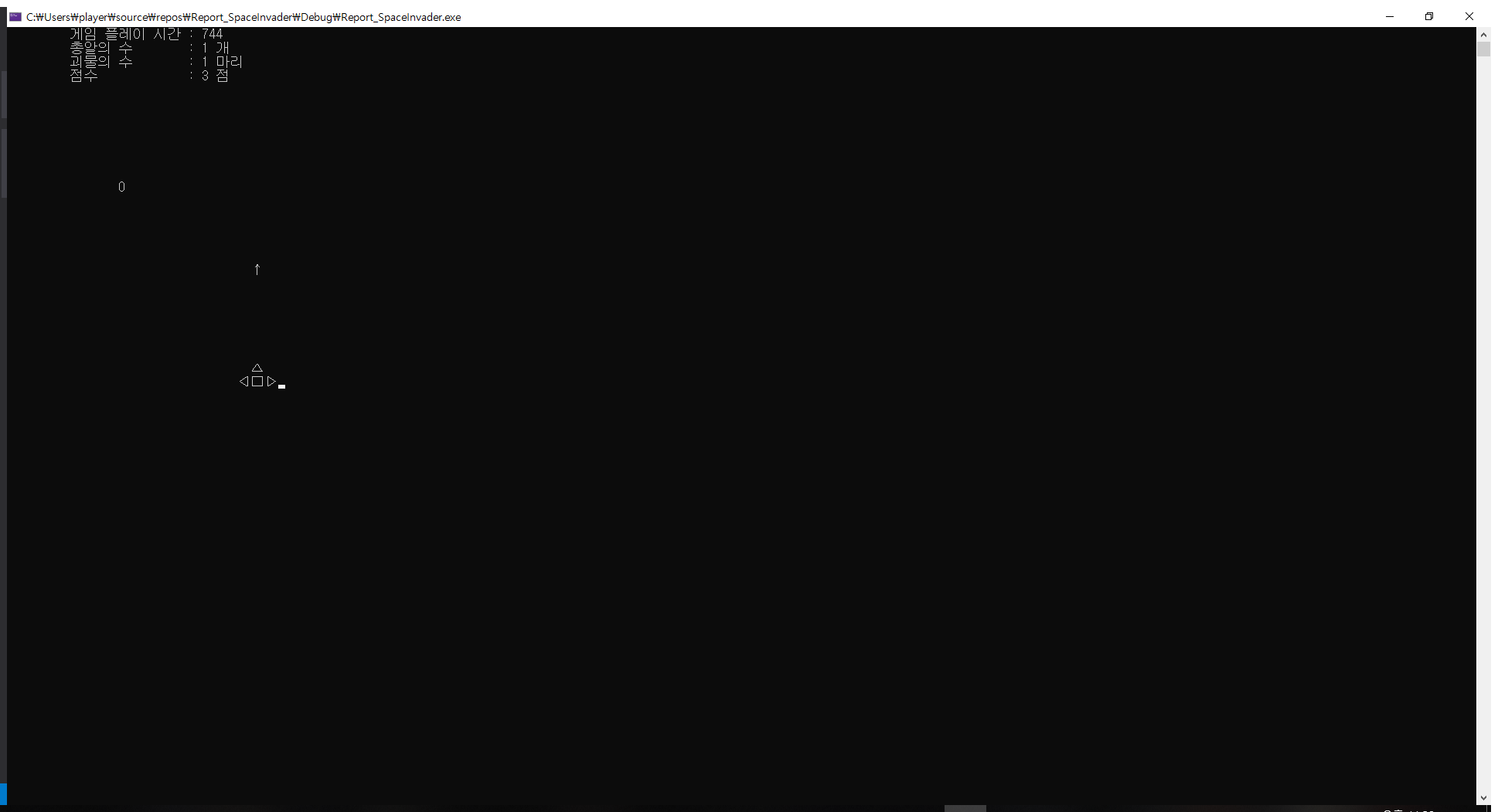
1,2,4 는 모두 앞선 function.cpp나 class.cpp로 구현이 되어 있어서 그것들을 그대로 가져다 사용을 하였으나, 3번은 조금 다르다. Switch 문을 바탕으로 해서 입력 받도록 설계를 하였는데, 제일 처음에 언급하였 듯이 프로그램이 동작하면서 계속 키보드의 입력을 받아야하는 상황이다. 그에 따라 while 반복문으로 프로그램이 계속 동작하도록 하였으며, 동작하는 과정 속에서 kbhit() 문을 이용해서 수시로 값을 읽어들이게 하였다. 이 함수는 키보드에 입력받은 값이 있으면 1을 반환하므로 만약 있다면 별도로 읽어오는 구문을 따로 설정해 주어야 한다. 그것이 바로 getch 함수이며, 이때 읽은 값을 바탕으로 방향키의 아스키 값을 찾아 이들을 이용해서 사용자의 우주선이 움직일 수 있도록 설계하였다.

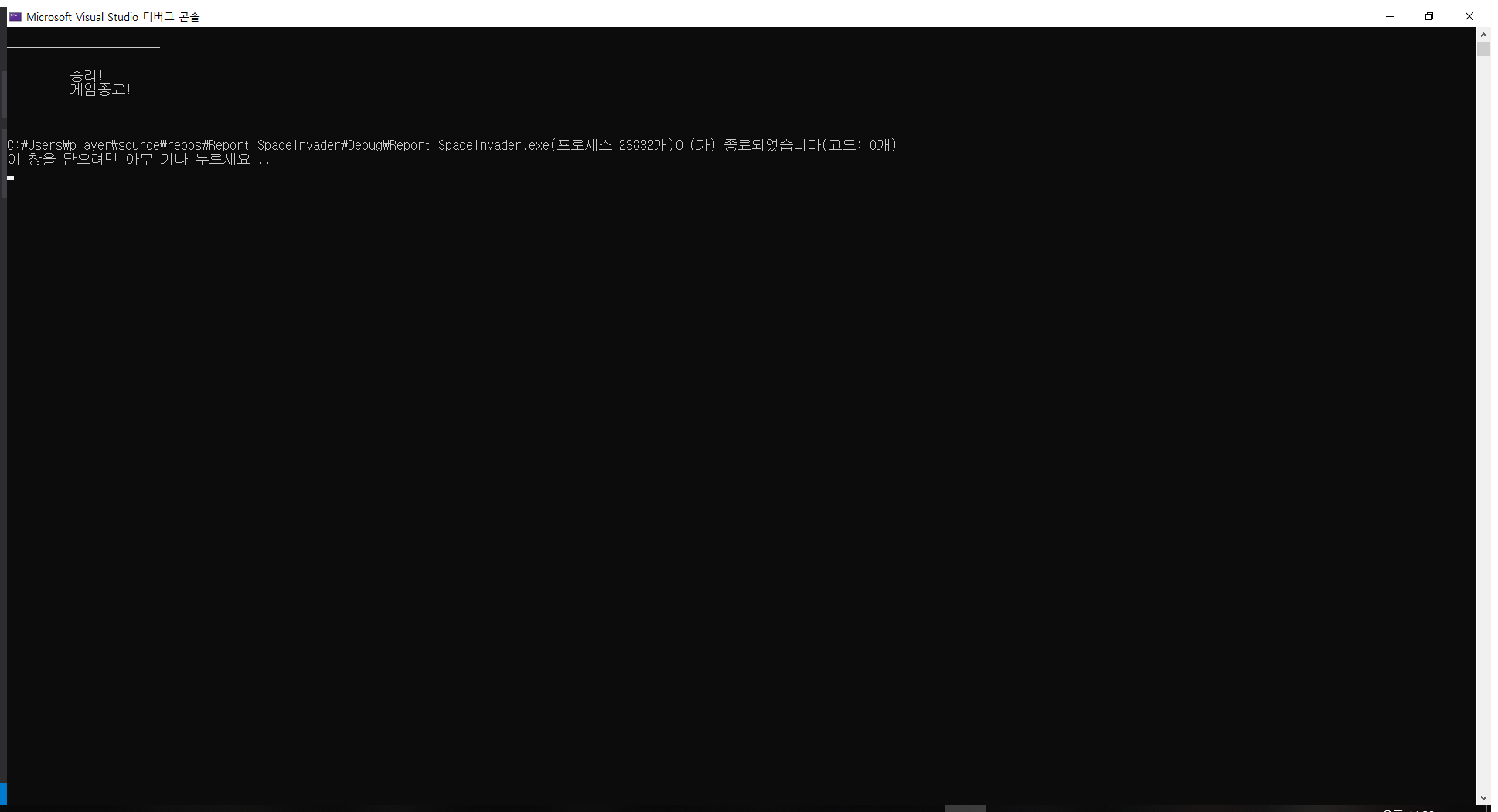
그리고 당연히 괴물이 모두 죽으면 승리이므로, 승리에 대한 부분과, 괴물이 플레이어에게 닿으면 플레이어가 지는 것이므로 패배 부분을 선언하여 승패 조건을 명확히 하였다.

**프로그램 작동 결과 및 분석과 후기.**









정상적으로 잘 작동하였다. 다만, 기본적인 UI가 콘솔창에서 모두 지워졌다가 출력되는 것인 만큼 약간의 버벅 거림은 존재하며, 멀티 스레딩을 기반으로 해서 제대로 따로 따로 동작하게 하는 것이 아닌 모두 순차적으로 동작이 되지만, 그 처리 연산이 무척이나 빨라 동시에 이뤄지는 것처럼 보이는 것이기에 잔 버그 등이 존재한다.

당초 계획서에는 몬스터는 ‘0’과 같은 형태가 아닌 다른 모양으로 구현하려고 하였지만, 좌표 단위로 인식을 하다 보니 bit 단위로 읽게 되어서 그 부분에서 인식 문제가 있어서 0으로 표기하였다. 만약, 범위를 지정해서 해당 범위에 속하는 영역에 총알이 닿는다면… 이라는 조건으로 진행을 하게 되면 버퍼문제가 발생하였고, 아직 버퍼에 대해서는 배우기 전이라 이에 대해 다루는게 미흡하기에 대처할 수 있는 능력이 부족하여 제외하였다.

기타 블록이나, 다른 부분들 역시 구현을 하게 되면 관련 되서 처리가 늘어나고 메모리에 올려야 하는 값들이 많아지는데 버퍼문제는 발생하지 않으나 연산이 많아진 탓인지 출력되는 게 끊기는 현상이 발생하였다.

그에 따라서 이들을 처리하는 방법을 찾아보았으나, 그렇게 하기 위해서는 이 프로그램의 작동 원리인 콘솔 위에 출력 후, 콘솔을 모두 지우고 다시 출력, 그리고 사용자가 보기에 매끄럽게 하기위해서 시간간격을 최소화. 라는 이 방식을 아예 뒤집어서 멀티스레딩이라는 개념 등을 도입해야 했기에 제외하였다. 그러다 보니 처음 계획안과는 많이 달라지게 되었다. 처음 만드는 프로그램인데, 스스로의 역량이 부족해서 구현하지 못한 것들이 많아 아쉬움이 남는다.

그럼에도 프로젝트를 진행하면서, 최대한 모든 것을 이해해보려고 노력을 하였고, 수업에서 배운 내용을 모두 활용해보려고 하였다. 그러다 보니, 나도 모르게 이들을 엮어서 이와 같은 프로그램을 만들 수 있었는데, 그 자체로 엄청나게 감격스러웠고, 어려웠던 전공이 한순간에 이해가 잘 되고 재미있어졌기에 자신의 역량에 대해 아쉬움이 남을 뿐, 되게 의미 있는 과제이자 활동이었다.

**깃허브 주소**

처음 사용하는 깃허브다 보니 꾸준하게 업로드를 하지 못하고, 이번 과제를 제출하면서 컴퓨터에서 계속 수정을 하다 최종 때에야 업로드를 하게 되었다.

프로그램은 총 5개가 올라가 있는데, main.cpp, function.cpp, class.cpp, Frame.h 와, 이들을 한데 묶어서 하나의 파일로 만든 backup.cpp가 존재한다.

과제에 제출하기 위한 것은 main.cpp, function.cpp, class.cpp, Frame.h 4개 이며, backup.cpp는 참고용이기에 앞의 4개만 링크만 포함한다.

깃허브: <https://github.com/SiE-Project/1st-Game>

Main.cpp: <https://github.com/SiE-Project/1st-Game/blob/master/main.cpp>

Function.cpp: <https://github.com/SiE-Project/1st-Game/blob/master/function.cpp>

Class.cpp: <https://github.com/SiE-Project/1st-Game/blob/master/class.cpp>

Frame.h: <https://github.com/SiE-Project/1st-Game/blob/master/Frame.h>