***Low-level***

**서버**

**SceneData 클래스, 구조체**

#define MAX\_PLAYER 3

#define MAP\_SIZE 30

#define BUF\_SIZE 512

~~#define BOMB\_EXPLOSION\_COUNT 3000000 폭탄처리를 서버에서 하기위한 카운트 추가함~~

서버시간 사용하면서 삭제

class SceneData // 게임에 관련된 데이터 관리, 업데이트

{

private:

PlayerStatus m\_playerStatus[MAX\_PLAYER];

std::vector<BombData> BombManger; //폭탄 관리용

int m\_nPlayer = 0; //char에서 int로 변경

public:

MapData m\_mapData[MAP\_SIZE][MAP\_SIZE];

private에서 public으로

void update();

void setKeyInput(SOCKET socket, KeyInput key);

~~void getMapData(MapData mapData[MAP\_SIZE][MAP\_SIZE]);~~

~~맵데이터 하나하나를 받아오게 수정함~~

MapData getMapData(int n, int m) { return m\_mapData[n][m]; };

void setPlayer(SOCKET socket);

void getPlayer(SOCKET socket, PlayerStatus \* playerStatus);

}

struct KeyInput //키 입력 on off

{

bool key\_Up;

bool key\_Down;

bool key\_Left;

bool key\_Right;

bool key\_Space;

};

struct Point

{

short X;

short Y;

};

enum Item

{

EMPTY,

BALLON, //폭탄 개수 증가

~~SHOES, //플레이어 이동 속도 증가~~

이동속도 없음

POTION, //폭탄 길이 증가

};

struct PlayerStatus //플레이어의 소켓, 물줄기, 이동속도, 위치, 상태를 저장

{

SOCKET playerSocket;

KeyInput key;

int bombCount; // 폭탄 개수

Point position; // 플레이어 위치

bool isAlive; // 생존 여부

int playerBombLength; //폭탄 불길 길이

PlayerColor playerColor;

bool isReady; // 준비 여부

};

Enum PlayerColor

{

EMPTY, //플레이어 없다

RED, //붉은색 플레이어 있다

GREEN, //초록색 플레이어 있다

BLUE, //파란색 플레이어 있다

};

struct MapData

{

Item item; //아이템이 있는지

bool isRock; //막혀있는 곳

bool isBomb; //폭탄이 있는지

bool isBombFrame; //폭탄 불길이 있는지

PlayerColor playerColor; //플레이어가 있는지

};

**서버, 클라이언트 통신**

DWORD WINAPI LobbyThread(LPVOID arg)

DWORD WINAPI GameThread(LPVOID arg) // 게임 로직 담당 쓰레드이다.

DWORD WINAPI ProcessThread(LPVOID arg)

// 각 클라이언트와 통신하여 키입력을 받고 맵 데이터를 넘겨준다.

HANDLE Event; //각 processthread는 하나가 동작하면 다른 쓰레드는 대기상태로

SceneData gameSceneData;

int buf[BUF\_SIZE];

int main()

{

Event //이벤트 생성

GThread //게임 쓰레드 생성

LThread //로비 쓰레드 생성

while (1)

{

printf("Running main \n");

Sleep(10000);

//실행중인거 확인

}  
}

DWORD WINAPI GameThread(LPVOID arg)

{

while (true)

{

gameSceneData.update();

}

return 0;

}

DWORD WINAPI LobbyThread(LPVOID arg) //클라이언트가 처음으로 연결되는 쓰레드

{

vector<SOCKET> MatchingQueue //연결된 소켓정보를 담음

if(MatchingQueue.size == MAX\_PLAYER)

{

Send //게임 시작한다는 참 값을 전달  
 CreateThread(NULL, 0, ProcessThread, MatchingQueue[i],0,NULL);

}

}

DWORD WINAPI ProcessThread(LPVOID arg)

{

SOCKET player\_sock = arg;

gameSceneData.setPlayer(player\_sock);

while(true)  
{

WaitForSingleObject(Event, INFINITE);

recv // KeyInput 클라이언트로부터 데이터 받기

gameSceneData.setKeyInput(player\_sock, input);

send // SceneData의 m\_mapData 를 클라이언트로 보내기

SetEvent(Event);

}

}

**클라이언트**

**SimpleGame**

GSEGame\* g\_game = NULL; //게임 씬 화면

GSELobby\* g\_lobby = NULL; //로비 씬 화면

KeyInput g\_inputs; //키 입력 저장

bool gameStart = false; //게임 시작 체크

void RenderScene(int temp) //타이머로 특정 프레임마다 불려질 함수

;

{

~~SendToServer(); //서버에 키 입력 전달하기~~

~~RecvFromServer(); //서버에서 맵데이터 전달받기~~

~~RenderScene에 합쳤다.~~

if (gameStart) //게임 플레이가 맞다면

{

g\_game->RendererScene(); //게임 화면 다시 그리기

}

Else

{

g\_lobby->RendererScene(); //대기씬 출력

}

glutSwapBuffers();

glutTimerFunc(16, RenderScene, 16);

}

Void KeyDownInput(unsigned char key, int x, int y); //키 눌렸을 시

Void KeyUpInput(unsigned char key, int x, int y); //키를 손에서 땟을 시

void SpecialKeyDownInput(int key, int x, int y); //방향키를 눌렀을 시

void SpecialKeyUpInput(int key, int x, int y); //방향키를 뗏을 시

11/19

~~void SendToServer() //서버에 키입력 전달하기~~

~~{~~

~~retval = send(sock, (const char\*)(&g\_inputs), sizeof(KeyInput), 0);~~

~~//고정 값 데이터 보내기~~

~~}~~

~~void RecvFromServer() //서버에서 맵데이터 받아오기~~

~~{~~

~~do{~~

~~retval = recvn(sock, reinterpret\_cast<char\*>(&mapData), sizeof(MapData), 0);~~

~~if (retval == SOCKET\_ERROR) {~~

~~err\_display("MapData recv()");~~

~~break;~~

~~}~~

~~} while (retval != 0);~~

~~g\_game->SetMapData(mapData); //받아온 맵 데이터 게임씬에 저장~~

~~}~~

~~Sendtoserver와 recvfromserver 제거하고 RenderScene에 합침~~

int connectSocket() //서버와 연결 함수

{

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)

return 1;

sock = socket(AF\_INET, SOCK\_STREAM, 0);

// connect()

ZeroMemory(&serveraddr, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, SERVERIP, &(serveraddr.sin\_addr.s\_addr));

serveraddr.sin\_port = htons(SERVERPORT);

retval = connect(sock, (SOCKADDR\*)&serveraddr, sizeof(serveraddr));

if (retval == SOCKET\_ERROR) err\_quit("connect()");

}

int main(int argc, char\* argv[])

{

//OpenGL설정

//서버와 연결

connectSocket();

g\_game = new GSEGame(); //게임씬 생성

glutTimerFunc(16, RenderScene, 16); //16msec마다 타이머 실행

//게임화면 갱신 장소

glutMainLoop();

}

**GSEGame**

**게임씬**

GSEGame::GSEGame(); //게임 씬 생성

void GSEGame::RendererScene(); //게임 화면 갱신

void GSEGame::SetMapData(MapData (\*map\_data)[MAP\_SIZE]); //MapData 게임씬에 저장

**GSELobby**

**로비씬**

GSELobby:: GSELobby (); //로비 씬 생성

void GSELobby::RendererScene(); //로비 화면 갱신

**Renderer**

**화면 출력 관련 함수**

Renderer::Renderer(int windowSizeX, int windowSizeY); //Renderer생성

void Renderer::Initialize(int windowSizeX, int windowSizeY); //화면설정 및 쉐이더연결

void Renderer::CreateVertexBufferObjects(); //VBO생성

void Renderer::AddShader(GLuint ShaderProgram, const char\* pShaderText, GLenum ShaderType);

//쉐이더 생성

bool Renderer::ReadFile(char\* filename, std::string\* target); //파일 읽기