***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

FinishGame finishgame; //게임 시작과 종료를 알릴 플래그 11/26 추가

bool startgame;

public:

std::vector<SOCKET> MatchingQueue;

//11/26 로비쓰레드에 필요한 정보를 씬데이터에서 관리

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]; }; 11/14 추가

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 // 칸마다 이동하게 하므로 이동속도 제거~~

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

};

**struct BombData //폭탄 타이머 및 폭탄 위치값 저장용 11/21 추가**

**{**

**Point bombPoint; //위치**

**std::chrono::system\_clock::time\_point bombCountdown; //타이머**

**int bombExplosionLength; //불꽃 길이**

**int playerID;**

**bool up = true;**

**bool down = true;**

**bool right = true;**

**bool left = true;**

**BombData(int px, int py, std::chrono::system\_clock::time\_point count, int length, int id) { bombPoint.X = px; bombPoint.Y = py; bombCountdown = count; bombExplosionLength = length; playerID = id; }**

**};**

**enum Winner 11/30 추가 승자를 클라측에 넘겨주기위해 추가**

**{**

**Winner\_RED,**

**Winner\_GREEN,**

**Winner\_BLUE,**

**Winner\_DRAW,**

**};**

**struct FinishGame 게임종료와 승자를 알리는 패킷**

**{**

**bool FinishGame;**

**Winner Winner;**

**};**

**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~~

~~}~~

**Event = CreateEvent(NULL, FALSE, TRUE, NULL);**

**if (Event == NULL) return 1;**

**GThread = CreateThread(NULL, 0, GameThread, NULL, 0, NULL);**

**if (GThread == NULL)**

**printf("Create GThread Error\n");**

**LThread = CreateThread(NULL, 0, LobbyThread, NULL, 0, NULL);**

**if(LThread == NULL)**

**printf("Create LThread Error\n");**

**while (1)**

**{**

**printf("Running main \n");**

**printf("startgame: %d\n", gameSceneData.MatchingQueue.size());**

**Sleep(10000);**

**if (gameSceneData.GetGameFinish().FinishGame) //게임이 끝날시 서버를 종료시킴**

**{**

**Sleep(5000);**

**printf("FinishGame: 1, ExitMain()\n");**

**return false;**

**}**

**}**}

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);~~

~~}~~

while (1) { **//11. 28변경**

addrlen = sizeof(clientaddr);

client\_sock = accept(listen\_sock, (SOCKADDR\*)&clientaddr, &addrlen);

gameSceneData.MatchingQueue.push\_back(client\_sock);

//클라이언트 소켓을 게임씬 데이터에서 관리

if (gameSceneData.MatchingQueue.size() == MAX\_PLAYER)

{

for (int i = 0; i < MAX\_PLAYER; ++i)

{

gameSceneData.SetPlayer(gameSceneData.MatchingQueue[i]);

//세명이 접속시 플레이어를 만들어줌

retval=send(gameSceneData.MatchingQueue[i],(char\*)&playerNum,sizeof(playerNum),0);

// 각 클라이언트에게 3명이 전부 접속함을 알림

PThread=CreateThread(NULL,0,ProcessThread,(LPVOID)gameSceneData.MatchingQueue[i],0, NULL);

//로세스 쓰레드를 생성

else

{

for (int i = 0; i < gameSceneData.MatchingQueue.size(); ++i)

{

retval=send(gameSceneData.MatchingQueue[i],(char\*)&playerNum,sizeof(playerNum), 0);

//3명이 아니면 접속한 클라이언트에게 몇 명이 접속한지 보내줌

}

}

} }

}

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

retval = send(client\_sock, (char\*)&finishFlag, sizeof(FinishGame), 0);

//게임 종료에 관한 정보도 클라이언트에게 보내줌 // 12.6 추가

SetEvent(Event);

}

}

**클라이언트**

**SimpleGame**

GSEGame\* g\_game = NULL; //게임 씬 화면

**(+)**GSELobby\* g\_lobby = NULL; //로비 씬 화면

**(+)**GSEFinishScene\* g\_finishscene = NULL; //게임 종료 화면

KeyInput g\_inputs; //키 입력 저장

~~bool gameStart = false; //게임 시작 체크~~

bool isPlay= true; //게임 종료 체크

void RenderScene(int temp) //타이머로 특정 시간마다 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);~~

if(!isPlay) //게임이 끝났으면

{

g\_finishscene->RendererScene(fg.Winner); //게임종료 화면으로 넘어가기

}

Else //게임이 안끝났으면

{

if (playerNum == 3) //플레이어 인원수가 3명일 경우

{

send(sock, (const char\*)(&g\_inputs), sizeof(g\_inputs), 0); //키입력 전달

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

//맵 데이터 받기

recvn(sock, reinterpret\_cast<char\*>(&fg), sizeof(fg), 0);

//게임 종료 여부와 승자 정보 받기

g\_game->SetMapData(mapData); //게임 씬 맵데이터 설정

g\_game->RendererScene(); //게임 씬 그리기

if (fg.FinishGame == 1) //게임이 끝났다면

{

isPlay = false;

}

}

else

{

recvn(sock, reinterpret\_cast<char\*>(&playerNum), sizeof(playerNum), 0);

//현 플레이어의 번호 받기

g\_lobby->RendererScene(playerNum); //로비 씬 출력

}

}

glutSwapBuffers(); //double buffering

glutTimerFunc(60, RenderScene, 60); //타이머 재귀

}

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

~~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

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

{

//OpenGL설정

//서버와 연결

connectSocket();

g\_game = new GSEGame(); //게임씬 생성

**(+)**g\_lobby = new GSELobby(); //로비씬 생성

**(+)**g\_finishscene = new GSEFinishScene(); //게임 종료씬 생성

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

glutTimerFunc(60, RenderScene, 60); //60msec마다 타이머 실행

//게임화면 갱신 장소

glutMainLoop();

}

**GSEGame**

**게임씬**

**(+)**MapData m\_pMapdata[MAP\_SIZE][MAP\_SIZE]; //맵데이터

**(+)**Renderer\* m\_renderer = NULL; //그리기 클래스

GSEGame::GSEGame(); //게임 씬 생성

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

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

**GSELobby**

**로비씬**

**(+)**int playerNumSave //플레이어 번호

**(+)**Renderer\* m\_renderer //그리기 클래스

GSELobby:: GSELobby (); //로비 씬 생성

void GSELobby::RendererScene(**(+)**int num); //로비 화면 갱신

**(+)GSEFinishScene**

**(+)게임 종료씬**

**(+)**Renderer\* m\_renderer //그리기 클래스

**(+)**void RendererScene(int num); //종료 화면 갱신

**(+)**void WinPlayer(int playerNum); //승리한 플레이어가 있을시

**(+)**void DrawPlayers(); //무승부일 시

**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

**(+)**void DrawSolidRect(float x, float y, float z, float sizeX, float sizeY, float r, float g, float b, float a);

//화면에 직사각형 그리기

**(+)**void Draw(~)(float x, float y); //(~)알파벳을 화면에 그리기