* High-Level
* Lobby

도표, 텍스트, 평면도, 기술 도면이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* In-Game

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* Low-Level
* Lobby Packet
  + C2S\_CREATE\_OR\_JOIN\_PACKET {

unsigned char packet\_size;

char packet\_type;

enum create\_or\_join;

};

* + C2S\_SELECTED\_PAGE\_PACKET {

unsigned char packet\_size;

char packet\_type;

unsigned short page\_number;

};

* + S2C\_LOBBY\_LIST\_PACKET {

unsigned char packet\_size;

char packet\_type;

unsigned short lobby\_count;

const LOBBY lobby\_list[LOBBY\_COUNT\_PER\_PAGE];

};

→ struct LOBBY {

unsigned short lobby\_id;

char lobby\_name[MAX\_LOBBY\_NAME\_LENGTH];

char current\_players;

};

* + C2S\_SELECTED\_LOBBY\_PACKET {

unsigned char packet\_size;

char packet\_type;

const LOBBY selected\_lobby;

};

* + S2C\_JOIN\_AVAILABILITY\_PACKET {

unsigned char packet\_size;

char packet\_type;

bool join\_availability;

};

* + S2C\_HOST\_ADDRESS\_PACKET {

unsigned char packet\_size;

char packet\_type;

const SOCKADDR host\_address;

};

* + H2S\_PLAYER\_DISCONNECTION\_PACKET {

unsigned char packet\_size;

char packet\_type;

};

* In-Game Packet
  + struct H2C\_PLAYER\_PACKET {

unsigned char packet\_size;

char packet\_type;

float x, y, z;

float dx, dy, dz;

char hp;

char animation\_state

char current\_element;

};

* + struct H2C\_ITEM\_PACKET {

unsigned char packet\_size;

char packet\_type;

char item\_type;

char item\_quantity;

};

* + struct H2C\_SKILL\_PACKET {

unsigned char packet\_size;

char packet\_type;

float x, y, z;

float dx, dy, dz;

char skill\_type;

bool destroyed;

};

* + struct H2C\_MONSTER\_PACKET {

unsigned char packet\_size;

char packet\_type;

float x, y, z;

float dx, dy, dz;

char hp;

char monster\_type;

char animation\_state;

};

* + struct C2H\_KEY\_ PACKET {

unsigned char packet\_size;

char packet\_type;

char key

}

* Server
  + int main() {

WSADATA WSAData;

WSAStartup(MAKEWORD(2, 2), &WSAData);

SOCKET s\_socket = WSASocket(…, WSA\_FLAG\_OVERLAPPED);

bind & listen;

HANDLE hIOCP = CreateIoCompletionPort(...);

CreateIoCompletionPort(s\_socket, hIOCP, ...);

do\_accept(...);

while (true) {

GetQueuedCompletionStatus(...);

switch (CompletionStatus) {

case ACCEPT:

WSARecv(c\_socket, ...);

do\_accept(...);

break;

case SEND:

switch (sent\_packet\_type) {

case S2C\_LOBBY\_LIST\_PACKET:

// Receive Selected Lobby

WSARecv(c\_socket, ...);

break;

case S2C\_JOIN\_AVAILABILITY\_PACKET:

if (can Join) {

S2C\_HOST\_ADDRESS\_PACKET p;

WSASend(c\_socket, ...);

}

else {

// Receive Selected Lobby

WSARecv(c\_socket, ...);

}

}

break;

case RECV:

switch (packet\_type) {

case C2S\_CREATE\_OR\_JOIN\_PACKET:

if (Create Lobby) {

Update Lobby List;

// Receive Player Disconnection Packet

WSARecv(c\_socket, ...);

}

else if (Join Lobby) {

S2C\_LOBBY\_LIST\_PACKET p;

WSASend(c\_scoket, ...);

}

break;

case C2S\_SELECTED\_LOBBY\_PACKET :

S2C\_JOIN\_AVAILABILITY\_PACKET p;

WSASend(c\_socket, ...);

break;

case H2S\_PLAYER\_DISCONNECTION\_PACKET:

Update Lobby List;

// Receive Player Disconnection Packet

WSARecv(c\_socket, ...);

break;

}

break;

}

}

closesocket(s\_socket);

WSACleanup();

}

* + void do\_accept(...) {

SOCKET c\_socket = WSASocket(…, WSA\_FLAG\_OVERLAPPED);

CreateIoCompletionPort(c\_socket, hIOCP, ...);

AcceptEx(s\_socket, c\_socket, ...);

}

* Client
  + int main() {

WSADATA WSAData;

WSAStartup(MAKEWORD(2, 2), &WSAData);

// Connect to Lobby Server

SOCKET s\_socket = WSASocket(…, WSA\_FLAG\_OVERLAPPED);

WSAConnect(...);

if (Create Lobby) {

// Send Lobby Creation to Server

C2S\_CREATE\_OR\_JOIN\_PACKET p;

WSASend(...);

Create Server Thread(server\_thread);

// Connect to Host

SOCKET h\_socket = WSASocket(…, WSA\_FLAG\_OVERLAPPED);

WSAConnect(…);

// Receive Game Data from Host

WSARecv(..., client\_recv\_callback);

Game\_Loop;

}

else if (Join Lobby) {

// Send Join Lobby to Server

C2S\_CREATE\_OR\_JOIN\_PACKET p;

WSASend(..., lobby\_send\_callback);

}

}

* + void CALLBACK lobby\_recv\_callback(...) {

switch (packet\_type) {

case S2C\_LOBBY\_LIST\_PACKET:

Show Lobby;

Select Lobby;

// Send Selected Lobby to Server

C2S\_SELECTED\_LOBBY\_PACKET p;

WSASend(..., lobby\_send\_callback);

break;

case S2C\_JOIN\_AVAILABILITY\_PACKET:

if (can Join) {

// Receive Host's Address from Server

WSARecv(..., lobby\_recv\_callback);

}

else {

Select Lobby;

// Send Selected Lobby to Server

C2S\_SELECTED\_LOBBY\_PACKET p;

WSASend(c\_socket, ..., lobby\_send\_callback);

}

break;

case S2C\_HOST\_ADDRESS\_PACKET:

// Disconnect from Lobby Server

closesocket(s\_socket);

// Connect to Host

SOCKET h\_socket = WSASocket(…, WSA\_FLAG\_OVERLAPPED);

WSAConnect(…);

// Receive Game Data from Host

WSARecv(..., client\_recv\_callback);

Start Game Loop;

break;

}

}

* + void CALLBACK lobby\_send\_callback(...) {

// Receive (Lobby List) or (whether can Join or not) from Server

WSARecv(..., lobby\_recv\_callback);

}

* + void server\_thread(...) {

Create Accept Thread(accept\_thread);

// Game Logic Loop

while (is\_game\_running) {

input\_queue.lock();

while (!input\_queue.empty())

Apply Input;

Input\_queue.unlock();

// Update Game Objects 60 Times per Second

if (Timeout) {

Game Logic;

// Send 20 ~ 30 Packets to Clients per Second

if (Timeout)

for (const auto& client : clients)

WSASend(...);

}

}

* + void accept\_thread(...) {

SOCKET s\_socket = WSASocket(…, WSA\_FLAG\_OVERLAPPED);

while (is\_game\_running) {

auto c\_socket = WSAAccept(s\_socket, ...);

// Receive Client's Key Input from Client

WSARecv(c\_socket, ..., host\_recv\_callback);

}

}

* + void CALLBACK host\_recv\_callback(...) {

if (Someone Disconnected) {

H2S\_PLAYER\_DISCONNECTION\_PACKET p;

WSASend(...);

}

Input\_queue.lock();i

input\_queue.push(key\_input);

input\_queue.unlock();

// Receive Client's Key Input from Client

WSARecv(c\_socket, ..., host\_recv\_callback);

}

* + Game Loop {

while (true) {

if (input)

// Send Client's Key Input to Host

WSASend(...);

// Update

while (!data\_queue.empty())

Apply Data;

Render;

}

}

* + void CALLBACK client\_recv\_callback(...) {

data\_queue.push(game\_data);

// Receive Game Data from Host

WSARecv(c\_socket, ..., client\_recv\_callback);

}

* 주차별 진행상황
* ~ 03.16(일) : 서버 구현에 필요한 과목 학습
* 03.17(월) ~ 03.23(일) : High-Level 구성
* 03.24(월) ~ 03.30(일) : IOCP 관련 내용 학습
* 03.31(월) ~ 04.15(화) : Low-Level 구성
* 중간 평가까지의 개발 일정
* 04.16(수) : Client, Server 공용 헤더 파일 작성
* ~~04.19(토) : Client 기초 틀 작성~~
* ~~04.20(일) : Client 中 server\_thread 함수 작성 (1)~~

04.20(일) : Client 기초 틀 작성

* ~~04.21(월) : Client 中 server\_thread 함수 작성 (2)~~

04.21(월) : Client 中 server\_thread 함수 작성

* ~~04.22(화) : Client 中 accept\_thread 함수 작성 (1)~~

04.22(화) : Client 中 accept\_thread 함수 작성

* ~~04.23(수) : Client 中 accept\_thread 함수 작성 (2)~~

04.23(수) : Client 中 Game Loop 작성

* ~~04.26(토) : Client 中 host\_recv\_callback 함수 작성~~

04.26(토) : Host와 Client 연결 단위 테스트

* ~~04.27(일) : Host와 Client 연결 단위 테스트~~
* ~~04.28(월) : Client 中 Game Loop 작성 (1)~~
* ~~04.29(화) : Client 中 Game Loop 작성 (2)~~
* ~~04.30(수) : Client 中 client\_recv\_callback 함수 작성 (1)~~

04.28(월) ~ 04.30(수) : Client 中 패킷 송수신 코드 작성

* ~~05.03(토) : Client 中 client\_recv\_callback 함수 작성 (2)~~

05.03(토) : Host와 Client 동기화 단위 테스트

* ~~05.04(일) : Host와 Client 동기화 단위 테스트~~
* 05.05(월) ~ 05.11(일) : 테스트 및 버그 수정
* 이후 일정
* 5월 : 로비 서버 구현
* 6월 : DB 연동 및 최적화
* 7월 : 테스트 및 버그 수정