**네트워크 게임 프로그래밍**

**추진 계획서**

2018180020 박재우

2020182031 이서연

PM – 2020182037 정롭비

**목차**

1. 애플리케이션 기획 ----------------------------------------------- **3p.**
2. High-Level 디자인 ----------------------------------------------- **5p.**
3. Low-Level 디자인 ----------------------------------------------- **~~8p.~~ 9p.**
4. 팀원 별 역할 분담 ----------------------------------------------- **~~11p.~~ 15p.**
5. 개발 환경 ------------------------------------------------------------ **~~12p.~~ 16p.**
6. 개발 일정 ------------------------------------------------------------ **~~13p.~~ 17p.**

**애플리케이션 기획**

컴퓨터 그래픽스 최종 프로젝트 / 2020182031 이서연, 2020182037 정롭비 기존 게임 제작

**게임 소개**

**장르** - 3인 3D 퍼즐게임

**컨셉** - 플레이어가 협동하여 별을 잡으면 클리어하는 게임

**예상 진행**

플레이어 각자 클라이언트에서 서버IP와 ID를 입력해 서버로 로그인한다. 최대 3인까지 접속 가능하다.

로그인이 성공하면 게임 로비로 들어오며 다른 플레이어를 확인할 수 있고 본인의 색상을 정해 게임 준비를 한다. 3명의 플레이어 모두 준비가 된다면 게임을 시작한다.

게임을 시작하면 각자 다른 위치에서 시작한다.

플레이어는 움직이고 점프할 수 있다.

맵의 아이템을 활용하여 누군가 별을 잡을 수 있도록 방법을 찾는다.

한 플레이어가 별을 잡는다면 다음 Scene으로 넘어가며 마지막 Scene이면 게임을 클리어 화면이 나타나고 잠시 뒤 게임 로비로 돌아온다.

게임 로비, 게임 진행을 하는 동안 채팅으로 소통할 수 있다.

하늘, 실외, 잔디, 일이(가) 표시된 사진

자동 생성된 설명텍스트, 잔디, 녹색, 실외이(가) 표시된 사진

자동 생성된 설명

└ 기존 게임 플레이 장면

**아이템**

곡괭이 - 금 간 블록을 부술 수 있고 블록을 생성할 수 있다.

신발 – 점프를 두 번 할 수 있다.

**조작법**

w/a/s/d – 플레이어 움직임

space – 플레이어 점프

r – 블록 설치 모드 (카메라 변환)

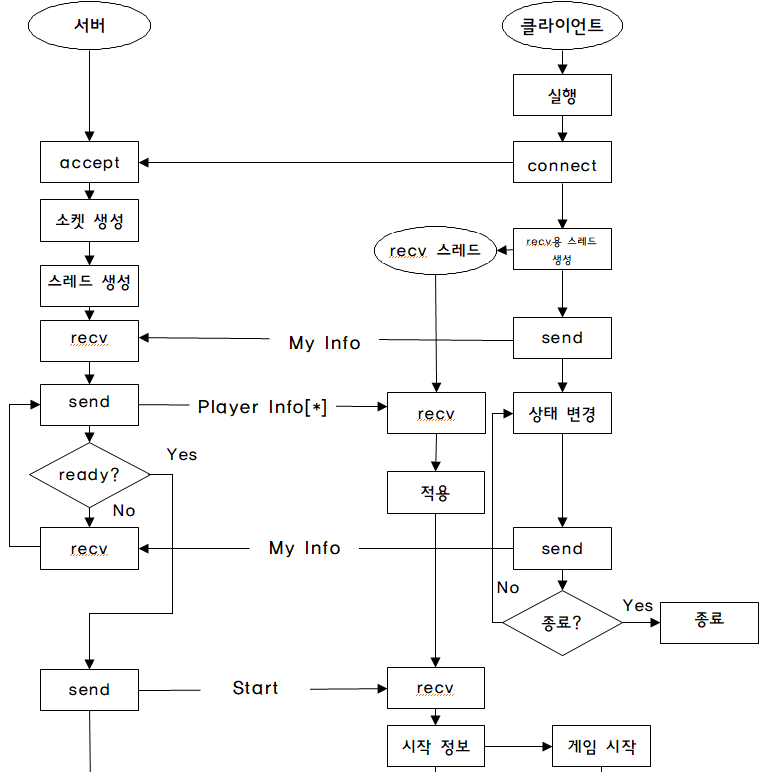
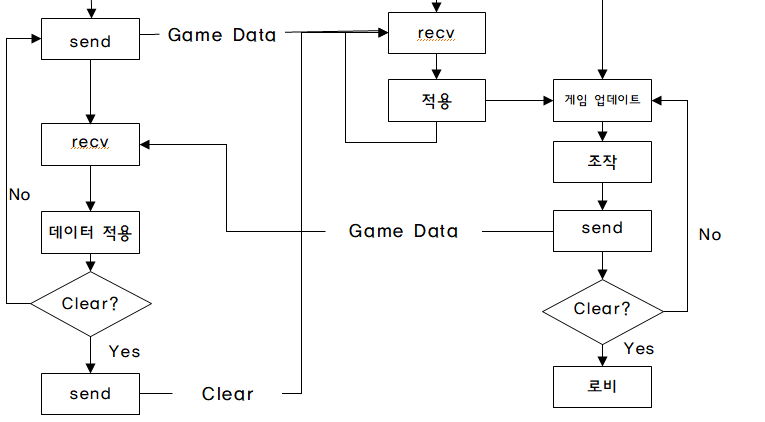
**High-Level 디자인**

**~~플로우 차트~~ 디자인 표**

테이블이(가) 표시된 사진

자동 생성된 설명

플로우 차트



1. **서버, 클라이언트 소켓 생성/연결**

**서버**

서버가 실행되면 윈속 초기화와 소켓 하나를 생성하고 그 소켓을 bind()로 서버의 대기 소켓으로 만든다. 그 후 클라이언트의 연결이 이루어지면 클라이언트마다 전용 소켓과 스레드를 할당한다.

**클라이언트**

클라이언트에서 서버의 IP, 본인의 ID를 입력하고 올바른 입력과 서버와 연결 할 수 있는 상태일 때 서버와 연결한다.

1. **게임 로비**

**서버**

각 클라이언트에게 저장되어 있는 모든 클라이언트 정보(ID, IP, Color, Ready 상태)를 보내고 클라이언트는 이 정보로 화면을 출력한다.

모든 클라이언트가 Ready 상태인 것을 확인하면, 서버는 게임 Scene 상태로 넘어가고 이 메시지를 클라이언트들에게 넘겨준다.

**클라이언트**

클릭 메시지(Ready 버튼)로 클라이언트 정보(ID, IP, Color, Ready 상태)를 서버로 보내 서버는 접속된 모든 클라이언트의 정보(ID, IP, Color, Ready 상태)를 저장한다.

서버에게 게임 Scene으로 전환되는 메시지를 받으면 게임 로비에서 Scene으로 전환한다.

1. **게임 Scene**

**서버**

클라이언트로부터 받은 각 플레이어들위 위치정보, 상호작용 블록 및 아이템의 충돌처리 결과, 설치된 블록의 좌표를 클라이언트들에게 넘겨준다.

충돌처리 결과를 보내주는 과정에서 ‘별’과 ‘플레이어’가 충돌 되었다면 다음 Scene으로 넘어가는 메시지를 클라이언트들에게 보낸다.

한 클라이언트 에게 게임 중단/재개 메시지를 받는다면 모든 클라이언트에게 게임 진행을 중단/재개하는 메시지를 클라이언트들에게 보낸다.

**클라이언트**

플레이어의 위치정보, 충돌처리 여부, 설치할 블록 좌표, 채팅을 서버로 넘겨준다.

서버로부터 게임 중단이나 재개 메시지를 받으면 그에 해당하는 게임 진행을 실행한다.

1. **게임 결과 및 종료**

**서버**

마지막 Scene에서 ‘별’과 ‘플레이어’가 충돌처리가 된다면 게임 클리어의 메시지를 클라이언트들에게 보내고 게임 로비 상태로 돌아간다.

게임 종료를 한 플레이어가 있다면 다른 플레이어에게 채팅으로 종료한 플레이어의 정보를 알려준다.

**클라이언트**

서버에서 게임 클리어 메시지를 받았다면 게임 로비 상태로 돌아간다.

게임 로비 상태에서 게임 종료 버튼을 누르면 서버와 연결을 끊으며 게임을 종료한다.

**Low\_Level 디자인**

class Game\_Data – 최상위 게임 정보

class Player\_Info\_Lobby : public Game\_Data – 로비에서 플레이어 정보

{

int Msg;// = MSG\_PLAYER\_INFO\_LOBBY;

char ID[10];

char ready\_state; // 0:white 1:R 2:G 3:B

char padding[34];

};

class Player\_Info\_Scene : public Game\_Data – 씬에서 플레이어 정보

{

int Msg;// = MSG\_PLAYER\_INFO\_SCENE;

Vector3 position; // 24byte

char ID[10];

char padding[24];

};

class Chat : public Game\_Data – 채팅 정보

{

int Msg;// = MSG\_CHAT;

char ID[10];

char Message[30];

char padding[6];

};

class Add\_Block : public Game\_Data – 블록 설치 정보

{

int Msg;// = MSG\_ADD\_BLOCK;

Vector3 position;

char padding[34];

};

class Collide : public Game\_Data – 충돌 정보

{

int Msg;// = MSG\_COLLIDE;

int item\_index;

char padding[42];

};

class Leave : public Game\_Data – 플레이어 게임 이탈 정보

{

int Msg;// = MSG\_LEAVE;

char ID[10];

char padding[36];

};

class Game\_Clear : public Game\_Data – 게임 클리어 정보

{

int Msg;// = MSG\_GAMECLEAR;

char padding[46];

};

class Pause : public Game\_Data – 일시 정지 정보

{

int Msg;// = MSG\_PAUSE;

bool pause;

char padding[44];

};

enum MSG – 데이터를 송수신 하기 전 어떤 메시지인지 구분하는 열거형 상수

{

MSG\_PLAYER\_INFO\_LOBBY – LOBBY에서의 플레이어 정보 메시지

MSG\_PLAYER\_INFO\_SCENE – SCENE에서의 플레이어 정보 메시지

MSG\_CHAT – 채팅 메시지

MSG\_ADD\_BLOCK – 블록을 설치하는 메시지

MSG\_COLLIDE – 충돌 메시지

MSG\_LEAVE – 플레이어가 종료한 것을 알려주는 메시지

MSG\_GAMECLEAR – 게임을 클리어한 것을 알려주는 메시지

MSG\_PAUSE – 게임 중단 메시지

~~MSG\_RESUME – 게임 재개 메시지~~ 제거

}

Class Object {

int id – 오브젝트마다 고유 ID

}

~~Class Player {~~

~~char ID[10], char color, char chatBuf[30] – 플레이어 ID, 플레이어 색상, 채팅 버퍼~~

~~}~~ 제거

**서버, 클라이언트 공통**

void sendMSG() – 메시지 정보를 보내주는 함수

MSG recvMSG() – 메시지 정보를 받아 해석하는 함수, 반환되는 값으로 서버와 클라이언트 모두 받은 메시지가 어떤 메시지인지 알 수 있다.

**클라이언트**

**송신**

void sendPlayerInfoLobby() – Lobby 상태일 때 ID, color 정보를 서버로 보내준다.

void sendPlayerInfoScene() - Scene 상태일 때 ID, Position 정보를 서버로 보내준다.

void sendChat() – chatBuf를 서버로 보내준다.

void sendCollide(int id) – 해당 id와 충돌되었다는 정보를 보내준다.

void sendAddBlock(float x, float z) – 블록을 설치할 위치를 서버에 보내준다.

**수신**

void recvPlayerInfoLobby() – Lobby 상태일 때 다른 플레이어의 ID, color 데이터를 받아온다.

void recvPlayerInfoScene() - Scene 상태일 때 다른 플레이어의 ID, Position 데이터를 받아온다.

void recvChat() – 채팅 메시지를 받아온다.

void recvCollide() – 충돌된 정보를 받아온다.

void recvAddBlock() – 블록 위치를 받고 떨어지는 블록을 설치한다.

void recvGameClear() – 게임을 클리어 했다는 메시지를 받는다.

void recvPlayerLeave(char\* id) – 해당 id를 가진 플레이어가 게임을 종료했다는 메시지를 받는다.

DWORD WINAPI **MsgRecvThread()** – 서버로부터 메시지를 받는 스레드

**서버**

**송신**

void sendPlayerInfoLobby() – Lobby 상태일 때 모든 클라이언트의 ID, color 정보를 클라이언트들에게 보내준다.

void sendPlayerInfoScene() - Scene 상태일 때 모든 클라이언트의 ID, Position 정보를 클라이언트들에게 보내준다.

void sendChat() – 채팅 메시지를 모든 클라이언트에게 보내준다.

void sendCollide(int id) – 입력된 id를 가진 오브젝트와 충돌된 것을 보내준다.

void sendAddBlock(float x, float z) – 모든 클라이언트에게 설치할 블록 위치를 보내준다.

void sendGameClear() – 게임을 클리어 했다는 메시지를 보내준다.

void sendPlayerLeave(char\* id) – 해당 id를 가진 플레이어가 게임을 종료했다는 메시지를 보내준다.

void sendPauseState() – 게임 중단/재개 메시지를 보내준다.

**수신**

void recvPlayerInfoLobby() – Lobby 상태일 때 ID, color 순으로 데이터를 받아온다.

void recvPlayerInfoScene() - Scene 상태일 때 ID, Position 데이터를 받아온다.

void recvChat() – 스레드마다 보낸 채팅 메시지를 받는다.

void recvCollide() – 충돌된 정보를 받는다.

void recvAddBlock() – 설치할 블록 위치를 받는다.

void recvPauseState() – 게임 중단/재개 메시지를 받는다.

동기화

DWORD WINAPI **ClientThread(**LPVOID arg) – 클라이언트와의 통신을 위한 스레드

arg에 클라이언트 SOCKET 포인터가 들어간다.

std::queue<GameData\*> MsgCommandQueue – 메시지를 임계영역을 사용하여 큐에 저장한다.

DWORD WINAPI Cacul\_Execute() – MsgCommandQueue에 저장된 메시지를 처리하는 스레드

클라이언트 스레드마다 처리해야 할 메시지를 하나의 큐에 저장하여 메시지에 대한 연산을 수행하고 클라이언트들에게 결과를 보내준다.

**팀원 별 역할 분담**

클라이언트 환경 구축 ------------------------------------------ ① 이서연 정롭비

기본 네트워크 환경 구축 (소켓/스레드 생성 등) ------------- ② 박재우 이서연 정롭비

메시지 식별 함수 ----------------------------------------------- ③ 정롭비

게임 로비

서버 클라이언트 정보 통신 환경 -------------------- ④ 박재우 이서연 정롭비

UI제작 ------------------------------------------------- ⑤ 이서연

게임 Scene

오브젝트 식별 ID 지정 ------------------------------- ⑥ 이서연

플레이어 Render 방식 설정 -------------------------- ⑦ 이서연 정롭비

플레이어 위치 및 충돌여부 데이터 send/recv ------ ⑧ 박재우 정롭비

키 입력 처리 send/recv ------------------------------ ⑨ 박재우 정롭비

게임 일시 중지 처리 --------------------------------- ⑩ 박재우 이서연 정롭비

채팅 --------------------------------------------------- ⑪ 박재우 이서연 정롭비

역할 표시 추가

~~박재우 – ②④⑧⑨⑩⑪~~

~~이서연 – ①②④⑤⑥⑦⑩⑪~~

~~정롭비 – ①②③④⑦⑧⑨⑩⑪~~ 제거

**개발환경**

운영체제 - Windows 10

프로토콜 - TCP/IP

IDE - Visual Studio 2022

언어 - C/C++ ws2\_32

라이브러리 - OpenGL

VCS - DVCS, Git 형상 관리

**개발일정(일별/개인별 계획 수립, 달력 형태로 작성)**

● 박재우 ● 이서연 ● 정롭비 식별색상 추가

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 11월 | | | | | | | |
| 일 | 월 | 화 | 수 | | 목 | 금 | 토 |
|  |  | 1 | 2 | | 3 | 4 | 5 |
|  |  |  |  | |  | 예비군 |  |
|  |  |  |  | |  |  |  |
|  |  |  |  | |  | 다른 플레이어 컨테이너 생성 | enum MSG,  sendMSG(),  recvMSG() 구현 |
| 6 | 7 | 8 | 9 | | 10 | 11 | 12 |
| 기본 네트워크 환경 구축 | | 서버, 클라이언트  sendPlayerInfoScene()  recvPlayerInfoScene(),  데이터 구조 클래스 구현 | | | | | sendPlayerLeave(),  recvPlayerLeave()  구현 |
|  | 로그인 UI 제작  로비 UI 제작 | | 클라이언트 오브젝트 식별 ID 지정 | | 데이터 구조 클래스 구현 | 서버, 클라이언트  sendPlayerInfoLobby()  ,recvPlayerInfoLobby()  구현 | |
| sendMSG(),  recvMSG() 구현 |  | 클라이언트 로그인 입력 버퍼 구현 | 클라이언트에서 입력한 정보로 서버와 연결 | | 데이터 구조 클래스 구현 |
| 13 | 14 | 15 | 16 | | 17 | 18 | 19 |
| sendPlayerLeave(),  recvPlayerLeave()  구현 | | |  | | MsgRecvThread() 구현 |  |  |
| 서버, 클라이언트  sendPlayerInfoLobby()  ,recvPlayerInfoLobby()  구현 | 로비 상태에서 접속된 플레이어 표시 | sendCollide(int id),  recvCollide()  구현 | | | | 플레이어 렌더링 텍스쳐링, 쉐이더 구현 | 디버그 및 테스트 |
| 클라이언트  모든 플레이어 렌더링 구현 |  |  | | MsgRecvThread() 구현 | 클라이언트recvCollide()의 결과 처리 |  |
| 20 | 21 | 22 | 23 | | 24 | 25 | 26 |
| 코드 병합 및 테스트 | | sendAddBlock(float x, float z), recvAddBlock() 구현 | | MsgCommandQueue 구조체 정의,  Cacul\_Execute() 구현,  ClientThread(LPVOID arg) 구현 | | |  |
| sendGameClear(),  recvGameClear()  구현 | |  |
| sendPauseState(),  recvPauseState()  구현 | | 클라이언트  recvAddBlock() 결과 처리 | |  |  |
| 27 | 28 | 29 | 30 | |
| sendChat(),  recvChat()  구현 | | |  | |
| 디버그 및 테스트 | | |  | |
| sendChat(),  recvChat()  구현 | | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 12월 | | | | | | |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
|  |  |  |  | 1 | 2 | 3 |
|  |  |  |  | 최종 테스트 및 보고서 준비 | | |
|  |  |  |  |
|  |  |  |  |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 최종 테스트 및 보고서 준비 | | | |