|  |  |  |  |
| --- | --- | --- | --- |
|  | 2020-11-02 |  | |
|  | |  | |
| 네트워크게임프로그래밍Term Project  2020-2 추진계획서 | | | |
|  |  | |  |
|  |  | | 게임공학부 2018180030 이동규 게임공학부 2018182030 이주호 |

목차

[<Slash Quest> 게임 개요 3](#_Toc55249462)

[설명 3](#_Toc55249463)

[상위 설계(High-Level Design) 5](#_Toc55249464)

[게임 흐름 5](#_Toc55249465)

[서버 / 클라이언트 동작 방식 6](#_Toc55249466)

[하위 설계(Low-Level Design) 8](#_Toc55249467)

[이동 데이터 패킷 8](#_Toc55249468)

[프로토콜 9](#_Toc55249469)

[클라이언트 함수 10](#_Toc55249470)

[bool SignIn(char\* id) 10](#_Toc55249471)

[bool RequestMatchMaking() 10](#_Toc55249472)

[void LoadMap() 10](#_Toc55249473)

[bool StartCountDown() 11](#_Toc55249474)

[void PlayerMove(Protocol type) 11](#_Toc55249475)

[void UpdatePlayers() 11](#_Toc55249476)

[void UpdateItemList() 11](#_Toc55249477)

[서버에서 사용하는 클라이언트 데이터 12](#_Toc55249478)

[서버 함수 13](#_Toc55249479)

[bool CheckUserID(const char\* id) 13](#_Toc55249480)

[void ProcessClient(Client\* arg) 13](#_Toc55249481)

[void StartMatchMaking() 13](#_Toc55249482)

[void InitPlayers(std::pair<PlayerPacket, PlayerPacket>) 13](#_Toc55249483)

[void CheckPlayersReady() 13](#_Toc55249484)

[bool Intersect() 13](#_Toc55249485)

[void CreateItem() 14](#_Toc55249486)

[void MatchDisconnect() 14](#_Toc55249487)

[개발 환경 및 역할 분담 15](#_Toc55249488)

[개발 환경 15](#_Toc55249489)

[역할 분담 15](#_Toc55249490)

[- 이동규 15](#_Toc55249491)

[- 이주호(Project Manager) 15](#_Toc55249492)

[개발 일정 계획 16](#_Toc55249493)

[개발일정 16](#_Toc55249494)

# <Slash Quest> 게임 개요

## 설명

프로젝트명: <Slash Quest>

Quarter-View 시점의 네트워크 대전 게임

플레이어가 자신의 주위를 회전하는 칼을 조작하여 상대방을 베는 2인 대전 게임이다.

플레이어는 메인 화면에서 Match-making을 시도하고, 매칭된 상대와 로비에 입장하여 게임을 시작한다.

커다란 마름모 모양의 플랫폼 위에서 게임을 시작하며, 일정 시간마다 랜덤하게 등장하는 아이템을 획득하거나, 플랫폼 위에 존재하는 ~~주위 오브젝트를 베면~~ 칼의 길이가 조금씩 늘어난다. 그와는 반대로 적 플레이어의 칼과 충돌할 경우 칼의 길이가 조금씩 감소한다.

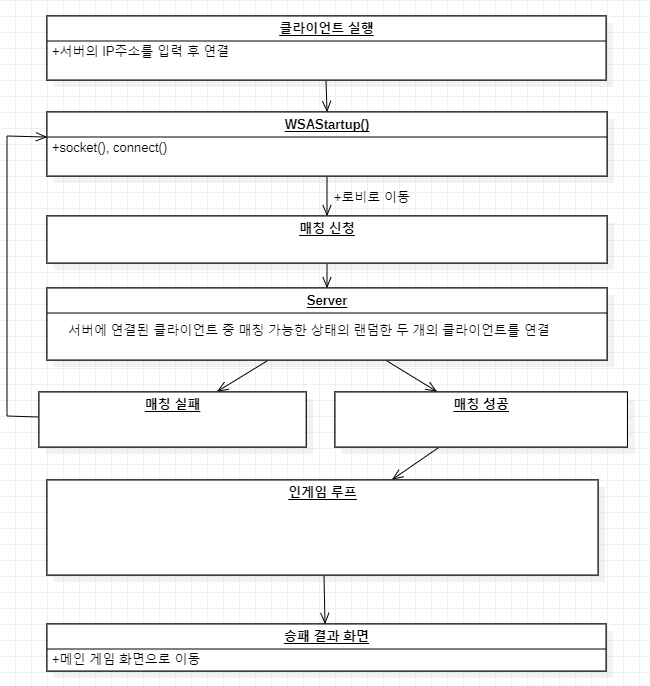
플레이어는 A, D 키를 눌러 각각 반시계 방향, 시계 방향으로 칼을 회전시킬 수 있으며, 두 키를 동시에 누르면 현재 칼이 가리키고 있는 방향으로 플레이어가 이동한다.

플레이어의 Hp는 5, 칼이 캐릭터와 충돌할 경우 해당 캐릭터의 Hp가 1 감소한다. 2명의 플레이어 중 먼저 Hp가 0이 되면 게임이 종료되며 각 플레이어의 승패를 확인할 수 있는 화면으로 이동한다. ~~플랫폼 밖으로 떨어진 플레이어가 발생할 경우 게임이 종료되며 해당 플레이어의 패배를 나타내는 게임 결과 화면으로 이동한다.~~

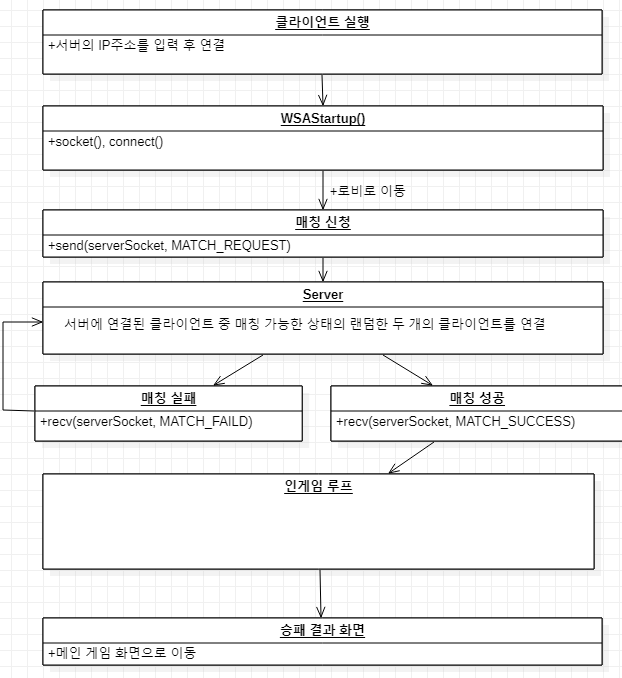
# 상위 설계(High-Level Design)

## 게임 흐름

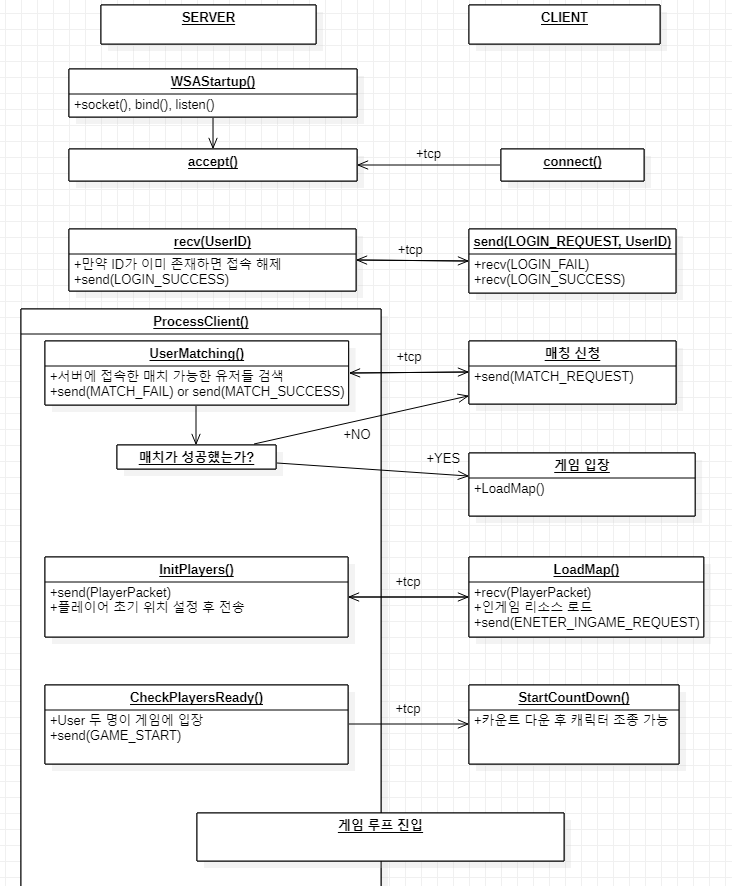
가정: 서버는 항상 켜져 있는 상태라고 생각한다.

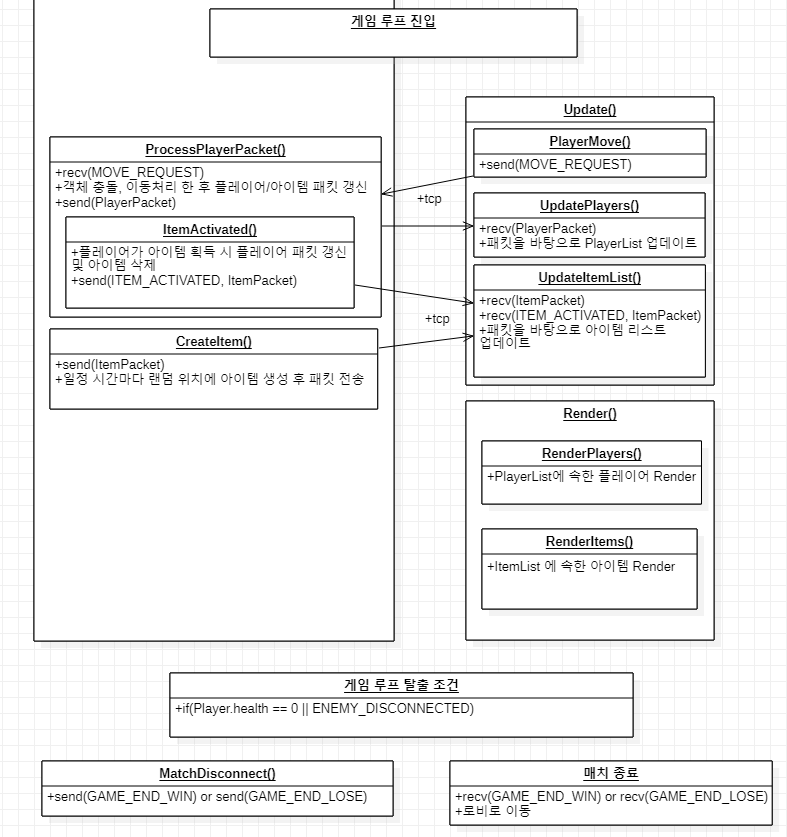


**--게임 흐름(수정)**

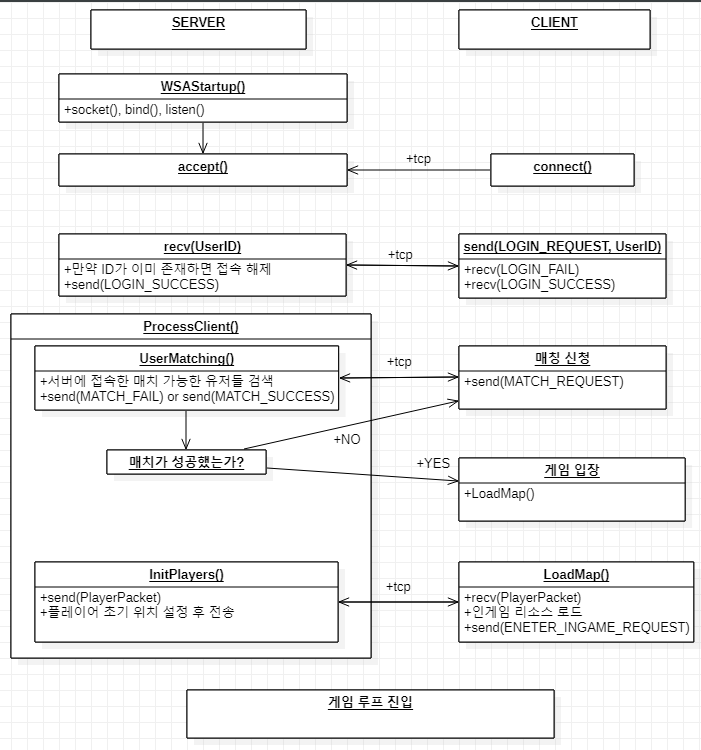


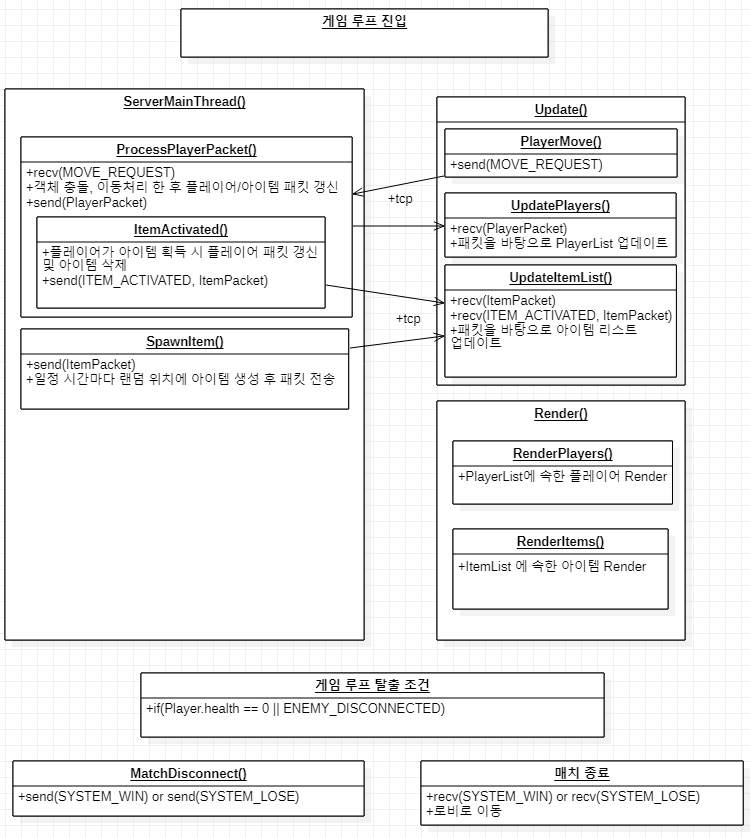
## 서버 / 클라이언트 동작 방식





**--서버/클라이언트 동작 방식(수정)**





# 하위 설계(Low-Level Design)

## 이동 데이터 패킷

struct Sword {

short length;

float rotation;

};

**Struct Packet{**

**float x, y;**

**}**

struct PlayerPacket **:** **public Packet** {

Sword sword;

unsigned char hp;

char id[MAX\_ID\_LEN];

bool isGodMode;

**stedy\_clock::time\_point time;**

};

struct ItemPacket **: public Packet** {

WORD type;

};

플레이어가 가지고 있는 칼은 플레이어를 기준으로 회전하기 때문에 위치에 관한 정보는 필요없이 칼의 길이와 플레이어 중심으로 회전각을 주고받는다. 실제로 보내는 데이터인 PlayerPacket안에 Sword구조체가 있고, PlayerPacket은 체력, 위치 좌표인 x와 y, 고유 키 값으로 ID, 그리고 현재 플레이어가 무적 상태인지를 판별하는 isGodMode가 있다. **무적 타이머를 계산하기 위해 시간 변수가 저장되어 있다**

ItemPacket은 아이템이 맵 상에 위치하게 될 위치 x, y와 아이템의 종류를 설명하는 WORD 타입의 변수가 있다.

## 프로토콜

클라이언트의 이벤트 발생을 서버에 알리는 프로토콜은 REQUEST로 끝나도록 하였고, 서버가 클라이언트에게 데이터를 전송할 때 사용되는 프로토콜은 데이터의 타입을 유추할 수 있는 이름으로 한다.

enum Protocol {

//LOGIN

LOGIN\_REQUEST = 100,

LOGIN\_FAIL,

LOGIN\_SUCCESS,

//LOBBY

MATCH\_REQUEST = 200,

MATCH\_FAIL,

MATCH\_SUCCESS,

//INGAME

~~ENTER\_INGAME\_REQUEST = 300~~,

~~GAME\_START,~~

PROCESS\_POINTS\_REAUEST,

KEYDOWN\_A\_REQUEST = 350,

KEYDOWN\_D\_REQUEST,

**KEYDOWN\_SUCCESS,**

MOVE\_REQUEST,

~~MOVE\_SUCCESS,~~

ITEM\_CREATED,

ITEM\_ACTIVATED,

ITEM\_BLIND\_ACTIVATED,

ITEM\_BLIND\_FINISHED,

//SYSTEM

ENEMY\_DISCONNECTED = 500

SYSTEM\_WIN,

SYSTEM\_LOSE,

};

## 클라이언트 함수

### bool SignIn(char\* id)

플레이어가 입력한 ID로 서버에 로그인 요청

send(LOGIN\_REQUEST, char\* id)  
ID를 서버에 전송, 서버는 중복된 ID가 있는지 검사

* + recv(LOGIN\_SUCCESS)  
    중복된 ID가 없다면 이 결과를 recv, 함수는 true 반환
  + recv(LOGIN\_FAIL)  
    중복된 ID가 있다면 이 결과를 recv, 함수는 false 반환

### bool RequestMatchMaking()

다른 유저와의 매칭 신청을 서버에 요청

send(MATCH\_REQUEST)  
서버에 매칭 신청에 해당하는 프로토콜 전송, 서버는 랜덤한 두 명의 매칭 가능한 유저 검색

* + recv(MATCH\_SUCCESS, pair<PlayerPacket, PlayerPacket>)  
    매칭이 성공하면 플레이어의 정보가 있는 pair와 프로토콜 recv, 함수는 true 반환
  + recv(MATCH\_FAIL, NULL)  
    매칭이 실패하면 이 결과를 recv, 함수는false반환

### void LoadMap(**SceneSequence sequence**)

매치 메이킹에서 받은 유저 정보를 바탕으로 Lobby에서 ~~In Game~~ **sequence**으로 씬 이동

### ~~bool StartCountDown()~~

~~두 클라이언트 모두 send(ENTER\_INGAME\_REQUEST)를 완료하면, 카운트다운 후 플레이어에게 이동 권한 부여~~

### void PlayerMove(Protocol type)

type에 따라서 서버에 어떤 키가 눌렸는지 전송한다.

send(KEYDOWN\_A\_REQUEST)  
서버에서 플레이어의 칼을 반시계로 회전시킨 후 양 쪽 클라이언트에 전송

send(KEYDOWN\_D\_REQUEST)  
서버에서 플레이어의 칼을 시계방향으로 회전시킨 후 양 쪽 클라이언트에 전송

send(MOVE\_REQUEST)  
서버에서 플레이어를 칼이 바라보는 방향으로 이동시킨 후 양 쪽 클라이언트에 전송

recv(MOVE\_SUCCESS, PlayerPackets)

### void UpdatePlayers(**PlayerPacket\* packet**)

PlayerMove(Protocol type)함수에서 받은 정보를 바탕으로 두 플레이어의 정보를 업데이트한다. 만약 Protocol type이 ENEMY\_DISCONNECTED이거나 한 플레이어의 체력이 0이면 게임 결과 Scene으로 이동한다.

### void UpdateItemList()

recv(ItemPacket)의 정보를 바탕으로 현재 게임 맵에 위치하는 아이템 정보를 업데이트한다.

## 서버에서 사용하는 클라이언트 데이터

struct GameClientInfo {

SOCKET socket;

PlayerPacket player;

Protocol protocol;

~~bool isMatching;~~

MatchingState matchState = MatchingState::LOBBY;

std::string name;

};

서버에서 저장하고 있는 클라이언트 구조체는 통신에 사용할 소켓과 플레이어 패킷, 프로토콜과 현재 클라이언트가 매칭중인 상태임을 저장하는 ~~bool변수~~ enum class 형의 MatchingState와 클라이언트의 이름을 저장한 string형의 name 변수가 있다.

## 서버 함수

### bool check\_id(const char\* id, GameClientInfo\* gp)

현재 서버에 접속한 유저들 중 id가 이미 존재하는지 검사해서 만약 이미 존재하는 id라면 false를 반환 후 클라이언트의 접속을 끊고, 사용 가능한 id라면 true를 반환하고 클라이언트 접속을 ~~허용한다.~~ 허용하고 해당 클라이언트의 이름을 설정해준다.

### DWORD WINAPI ProcessClient(Client\* arg)

서버에 접속한 클라이언트를 대상으로 스레드를 생성하고 해당 클라이언트에서 보내는 데이터를 처리하는 스레드 함수이다.

### void StartMatchMaking()

이 함수에 진입할 때 EnterCriticalSection()을 호출하고 서버에 접속한 유저들 중 매치 메이킹을 요청한 클라이언트 두 개를 pair로 묶어 vector에 저장한 후 함수를 종료하기 직전에 LeaveCriticalSection()을 호출한다.

### ~~void InitPlayers(std::pair<PlayerPacket, PlayerPacket>)~~

void InitPlayers(PlayerPacket\*)

~~매치가 성사된 두 클라이언트의 게임 내 초기 위치를 설정하고 양쪽 클라이언트에 send(MATCH\_SUCCESS, pair<PlayerPacket, PlayerPacket>)를 송신한다.~~

전달된 플레이어의 초기 위치를 설정한다.

### ~~void CheckPlayersReady()~~

~~게임에 입장한 유저 두 명이 모두 ENTER\_GAME\_REQUEST를 송신했다면, 서버는 접속을 확인하고 GAME\_START를 송신한다.~~

### ~~bool Intersect~~ void CheckCollision(\*)(**Packet\*, Packet\***)

플레이어가 아이템이나 상대 플레이어의 칼과 충돌하면 정보를 갱신하고true를 반환한다. 충돌이 발생하지 않았다면false를 반환한다.

### void SpawnItem()

맵 내에 랜덤한 위치에 아이템을 생성 후 클라이언트에 아이템의 정보를 전달한다. send(ITEM\_CREATED, ItemPacket)  
서버에서 아이템이 생성되었음을 알리는 프로토콜과 아이템의 위치, 타입 등을 담고 있는Packet을 전송한다.

### void MatchDisconnect()

어느 한 플레이어의 체력이 0이거나 연결이 끊겼을 경우 연결이 지속 중인 클라이언트에 send(ENEMY\_DISCONNECTED)하고, 매칭 플레이어를 관리하는 vector에서 제거한다.

**--추가된 함수**

DWORD WINAPI ServerMainThread(LPVOID arg)

현재 인게임에서 매치가 성사된 두 플레이어를 묶어서 아이템 생성이나 이동에 관한 이벤트를 지속적으로 업데이트 시켜준 후에 두 클라이언트에 전송한다.

# 개발 환경 및 역할 분담

## 개발 환경

* + 운영체제: Windows 10
  + 통신 프로토콜: TCP/IP
  + 통합 개발 환경(IDE): Visual Studio 2019
  + 사용 API: Win32

## 역할 분담

### 이동규

* + Server: 네트워크 통신, 충돌 연산 처리
  + Client: 게임 루프 내 Update, Render 함수에 필요한 요소 구현

### 이주호(Project Manager)

* + Server: 멀티 스레드, 네트워크 통신, 통신 관련 클래스 구현
  + Client: 게임 프레임워크 설계, Scene 설계 및 관리

# 개발 일정 계획

## 개발일정

|  |  |  |
| --- | --- | --- |
|  | 이동규 | 이주호 |
| 11/6 금 |  | 클라이언트 프레임워크 설계 |
| 11/7 토 | 기존 코드 자원 관리 방식 변경  기존 코드 함수 이름 / 호출 방식 변경 |  |
| 11/8 일 | 기존 코드 자원 관리 방식 변경  기존 코드 함수 이름 / 호출 방식 변경 |  |
| 11/9 월 |  | 클라이언트SceneManager 기초 설계 |
| 11/10 화 |  | 클라이언트SceneManager 구현 |
| 11/11 수 |  |  |
| 11/12 목 |  |  |
| 11/13 금 | 대전게임에 불필요한 콘텐츠 제거와  그에 따른 기존 코드 구조 변경 | SceneManager 멤버 함수 및  Scene 이동 확인 작업 |
| 11/14 토 | ~~클라이언트 – 서버 간 이동 처리 송수신 작업~~ (11/24 로 변경) | 서버 프레임워크 구현 |
| 11/15 일 | ~~클라이언트 – 서버 간 이동 처리 송수신 작업~~  (11/25 로 변경) |  |
| 11/16 월 | ~~충돌 처리 구현~~ (11/30으로 변경) | ~~플레이어 접속 및 ID확인~~ |
| 11/17 화 | 클라이언트 – 서버 간 ID,  플레이어 정보 송수신 작업 |  |
| 11/18 수 | ~~충돌 처리 구현~~ |  |
| 11/19 목 |  |  |
| 11/20 금 | ~~클라이언트 – 서버 간 충돌처리 송수신 작업~~ | ~~아이템 관련 구조체, 함수 구현~~ |
| 11/21 토 | ~~클라이언트 – 서버 간 충돌처리 송수신 작업~~ | ~~아이템 랜덤 생성 및 패킷 이동~~ |
| 11/22 일 |  |  |
| 11/23 월 |  | ~~Match-Making 자료구조, 함수 구현~~ |
| 11/24 화 | 클라이언트 - 서버 간 이동처리 송수신 | 클라이언트 – 서버 연결,  플레이어 접속 및 ID확인 |
| 11/25 수 | 클라이언트 - 서버 간 이동처리 송수신 ~~아이템 효과 구현~~ (11/27로 변경) |  |
| 11/26 목 | 플레이어 패킷 통신 (+추가 기록) ~~아이템 효과 구현~~ | ~~아이템 관련 구조체, 함수 구현~~ |
| 11/27 금 | ~~아이템 효과 구현,~~  ~~플레이어 패킷 통신~~ | ~~아이템 랜덤 생성 및 패킷 이동,  플레이어 패킷 통신~~ |
| 11/28 토 | 로비 및 메인 화면 구현 |  |
| 11/29 일 | 로비 및 메인 화면 구현  (이주호가 구현) |  |
| 11/30 월 | ~~충돌처리~~  (12/3 최종 수정 예정) | ~~임계영역 설정, 통신 확인~~ |
| 12/1 화 |  | ~~멀티 스레드 동기화~~ |
| 12/2 수 | ~~클라이언트 – 서버 간 ID,~~  ~~플레이어 정보 송수신 작업~~ |  |
| 12/3 목 | ~~충돌처리~~ |  |
| 12/4 금 |  | ~~경우에 따른 승패 처리 및~~  ~~매칭 해제~~  (12/6로 변경) |
| 12/5 토 | ~~버그 수정 & 최종 빌드~~ | ~~Bug fix, 최종 빌드~~ |
| 12/6 일 | ~~서버 구조 변경(추가)~~ | ~~경우에 따른 승패 처리 및~~  ~~매칭 해제~~ |
| 12/7 월 | ~~프로젝트 마감~~ | ~~프로젝트 마감~~ |
| 12/8  화 |  |  |
| 12/9  수 | Remote 테스트 최종 검수(추가) | Remote 테스트 최종 검수(추가) |