네트워크 게임 프로그래밍

Term Project 추진계획서_v3

3팀

팀장 2021184025 이도익

팀원 2019180037 장윤서

팀원 2021184033 조성욱

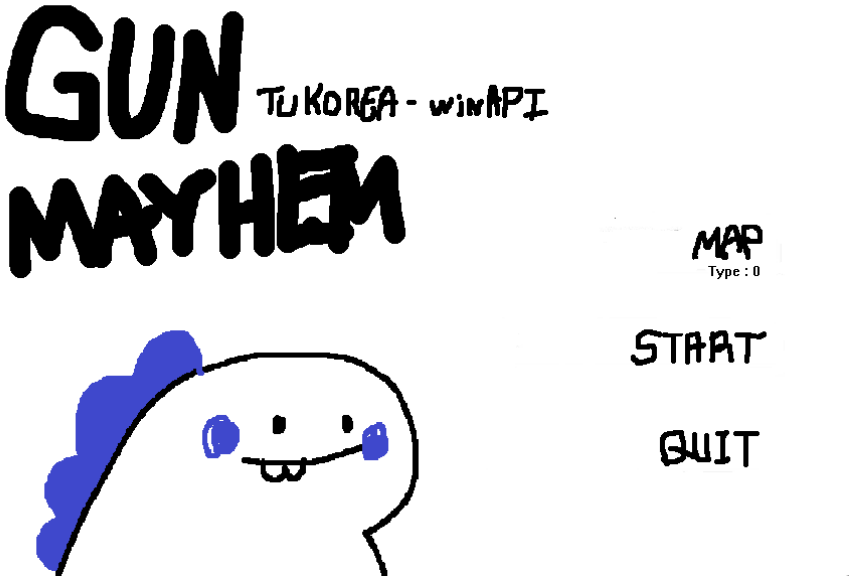
목차

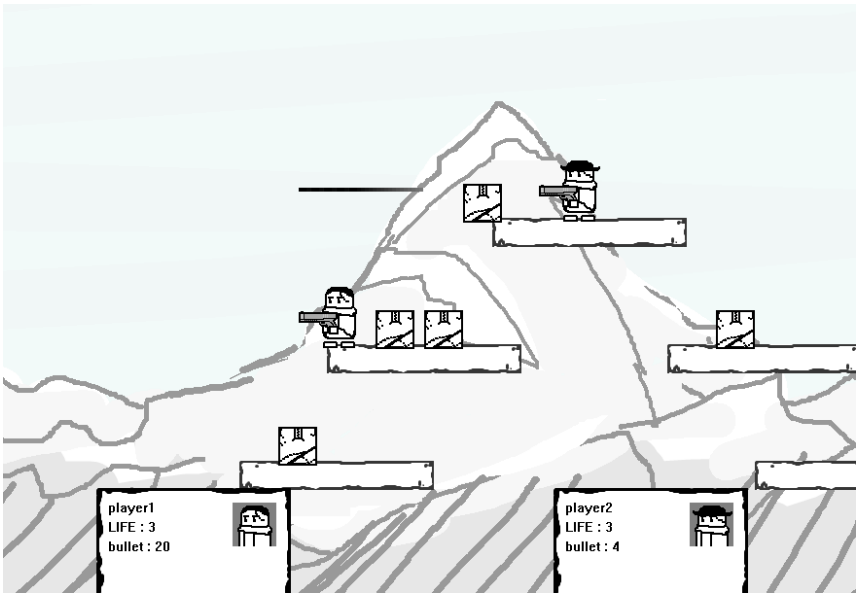
게임 개요

2D PVP 슈팅 게임인 "Gun Mayhem"이라는 게임을 모작한 게임.

총을 적중 시켜 적을 밀어내어 적이 맵 밖으로 이탈하면 사망한다.

적을 일정횟수 이상 탈락(사망)시키면 승리하는 게임이다.





서버 개요

- 기존 단일 클라이언트 환경에서 동작하던 게임에 서버를 도입하여 멀티플레이 1:1 환경을 구현한다.

- 플레이어의 위치, 점수, HP 등 게임 데이터를 실시간으로 동기화하며, 클라이언트 간 상태 일관성을 보장한다.

- 서버에서 게임 월드에 대한 모든 연산처리를 수행하며, 각 클라이언트는 서버에 자신의 Input 값 만을 송신한다.

- 서버는 자원 관리를 수행하며, 게임 종료 시 스레드, 소켓, 메모리를 안전하게 정리한다.

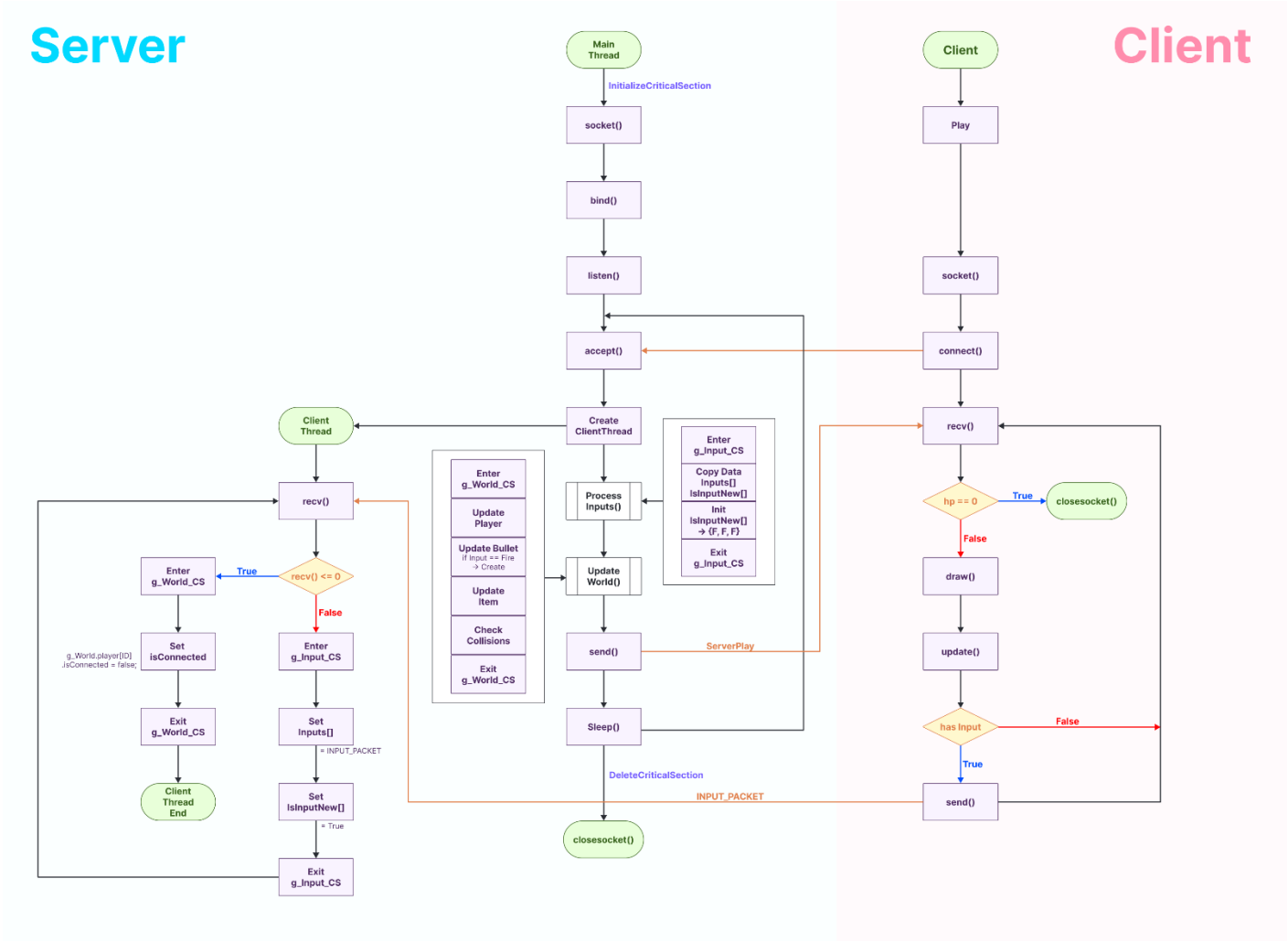
- 서버는 클라이언트가 직접 서로 통신하지 않아도 되도록 중계 역할을 수행하며, 게임의 중심적인 역할을 맡는다.

< 추가사항 >

- 서버는 각 게임룸별로 독립적인 스레드를 운영하여, 대기 상태와 게임 진행 상태를 관리한다.

- 서버 중심 구조를 통해 동시에 여러 게임룸을 운영할 수 있으며, 각 방은 서로 영향을 받지 않고 독립적으로 동작한다.

하이 레벨 디자인


Server
Client
Main Thread
InitializeCriticalSection
socket()
bind()
listen()
accept()
Create ClientThread
Client Thread
recv()
Enter g_World_CS
Update Player
Update Bullet
If Input == Fire
-> Create
Update Item
Check Collisions
Exit g_World_CS
Process Inputs()
Enter g_Input_CS
Copy Data Inputs[] IsInputNew[]
Init IsInputNew[] -> {F, F, F}
Exit g_Input_CS
Update World()
send()
ServerPlay
Sleep()
DeleteCriticalSection
closesocket()
recv() <= 0
True
False
Enter g_World_CS
Set isConnected
g_World.player[ID].IsConnected = false;
Exit g_World_CS
Client Thread End
Enter g_Input_CS
Set Inputs[]
= INPUT_PACKET
Set IsInputNew[]
= True
Exit g_Input_CS
INPUT_PACKET
Client
Play
socket()
connect()
recv()
hp == 0
True
closesocket()
False
draw()
update()
has Input
False
True
send()

로우 레벨 디자인 – struct

서버 – 클라이언트 공용 구조체

```
enum ITEMTYPE { ITEM_NONE, ITEM_PISTOL, ITEM_SNIPER };
```

[ITEMTYPE] 아이템 종류

```
enum PLAYERANIM {
PLAYER_NONE,
PLAYER_STAND_LEFT,
PLAYER_STAND_RIGHT,
PLAYER_WALK_LEFT,
PLAYER_WALK_RIGHT
};
```

[PLAYERANIM] 플레이어 애니메이션

```
enum ACTION {
ACTION_NONE,
ACTION_MOVE_R,
ACTION_MOVE_L,
ACTION_JUMP_UP,
ACTION_JUMP_DOWN,
ACTION_FIRE
}
```

[ACTION] 플레이어 키 입력

```
Struct PlayerStatus {
Vec2            vPos;
PLAYERANIM      eAnim;
ITEMTYPE        eTYPE = ITEM_NONE;
Int             iLife = 0;
Vec2            vHit;
Bool            isConnected = false;
ACTION          eAction = ACTION_NONE;
};
```

[PlayerStatus] 플레이어 상태

```
struct Bullet {
    vec2        vPos;               // 현재 위치값
    vec2        vDir;               // 나아가는 방향
    float       fDis = 0.f;         // 남은 사거리
    bool        isDead = false;     // 삭제 판정
};
```

[Bullet] 총알 구조체

```
struct ItemBox {
    vec2        vPos;                   // 현재 위치값
    vec2        vDir;                   // 나아가는 방향
    ITEMTYPE    eType = ITEM_NONE;      // 안에있는 아이템 타입
    bool        isDead = false;         // 삭제 판정
};
```

[ItemBox] 아이템박스 구조체

로우 레벨 디자인 – struct

서버

```
struct ServerPlay {
    PlayerStatus            status;                 // 상대 캐릭터 상태
    size_t                  iBulletNum = 0;         // 모든 총알 개수
    size_t                  iItemBoxNum = 0;        // 모든 아이템 박스 개수
    std::vector<Bullet>     vecBullet;              // 모든 총알 정보 벡터
    std::vector<ItemBox>    vecItemBox;             // 모든 아이템 박스 정보 벡터
    vec2                    vHit;                   // 기본으로 0.f, 0.f / 총알과 충돌 시 총알의 방향 벡터
    ITEMTYPE                eItemType = ITEM_NONE; // 변경할 아이템 타입
};
```

↓ 수정사항

```
Struct ServerPlay {
 PlayerStatus                status[3];
 Size_t                      iBulletNum = 0;
 Size_t                      iItemBoxNum = 0;
 Std::vector<Bullet>         vecBullet;
 Std::vector<ItemBox>        vecItemBox;
};
```

[ServerPlay]

게임 플레이 레벨에서 서버가 클라이언트에 전송할 구조체.

클라이언트

```
struct ClientPlay {
    PlayerStatus            status;             // 내 캐릭터 상태
    size_t                  BulletNum = 0;      // 이번 프레임에 생성한 총알 개수
    std::vector<Bullet>     vecBullet;          // 이번 프레임에 생성한 총알 정보 벡터
};
```

↓ 수정사항

```
Struct ClientPlay {
 ACTION              eAction = ACTION_NONE;
};
```

[ClientPlay]

게임 플레이 레벨에서 클라이언트가 서버에 전송할 구조체.

로우 레벨 디자인 – Function

```
void Update() {
  DeleteObject() // 총알과 아이템 박스 isDead 인 것 삭제 처리 함수
  UpdateBullet() // 모든 총알 위치 업데이트 함수
  UpdateItemBox() // 모든 아이템 박스 위치 업데이트 함수
  CollisionPlayer() // 플레이어와 총알, 아이템 박스 충돌 처리 함수
}
```

Update() 추가사항

UpdatePlayer()

// 플레이어 인풋에 대한 위치 변경 함수

```
void DeleteObject() {
총알과 아이템박스 벡터를 순회하며 isDead가 true인것을 삭제한다
}
```

```
void UpdateBullet() {
총알 방향값과 속도로 현재 위치와 남은 사거리를 수정한다
사거리가 0이 되면 isDead를 true로 바꾼다
}
```

```
void UpdateItemBox() {
아이템 박스 방향값과 속도로 현재 위치를 수정한다
아이템 박스의 속도가 0이 아니라면 맵 블록과 충돌 처리를 하고 충돌 시 속도를 0으로 바꾼다
}
```

```
void CollisionPlayer() {
플레이어와 총알의 충돌 처리를 진행하고 충돌 시 vHit에 총알의 vDir을 넣어주고 총알의 isDead를 true로 바꾼다
플레이어와 아이템 박스의 충돌 처리를 진행하고 충돌 시 eItemType을 아이템 박스의 eType을 넣어주고 아이템 박스의 isDead를 true로 바꾼다
}
```

로우 레벨 디자인 – 스레드 및 동기화

메인 스레드

역할: 클라이언트 접속 관리 (accept( ))

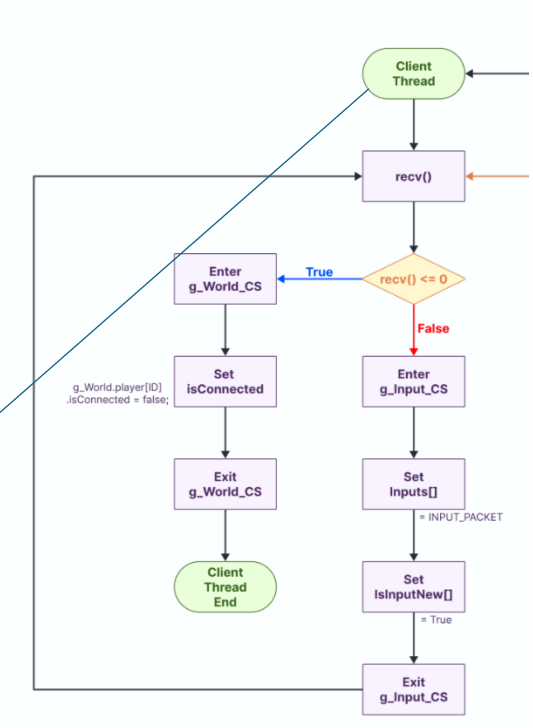
- 클라이언트 접속 요청 수락
- 클라이언트별 스레드 생성
- 게임 로직 및 통신 처리 (send( ))

```
MainThread
{
    InitalizeCriticalSection()
    socket()
    bind()
    listen()
    while (1) {
        accept()
        CreateThread(ClientThread)
        Update()
        send()
    }
    DeleteCriticalSection()
    closesocket()
}
```

```
Update() {
EnterCriticalSection()
UpdatePlayer()
UpdateBullet()
UpdateItem()
CheckCollisions()
LeaveCriticalSection()
}
```

클라이언트 스레드

- 클라이언트 통신 처리 (recv( ))
- 클라이언트 접속 종료 판단
- 클라이언트가 3 명 미만일 때 : " 대기 중 " 상태 유지
- 클라이언트가 3 명 모이면 : 즉시 " 게임 시작 "




```
ClientThread() {
    recv()
    EnterCriticalSection()
    InsertInput()
    LeaveCriticalSection()
}
```

(수정)로우 레벨 디자인 – 스레드 및 동기화

서버 - 메인 스레드

```
MainThread
{
    InitalizeCriticalSection()
    socket()
    bind()
    listen()
    CreateThread(AcceptThread)
        CreateThread(ProcessClient)
    while (!isGameEnd) {

        EnterCriticalSection()
        // 키 입력 처리
        LeaveCriticalSection()

        EnterCriticalSection()
        UpdatePlayer()
        UpdateItemBoxes()
        UpdateBullets()
        Collision();
        send()
        LeaveCriticalSection()
    }
    DeleteCriticalSection()
    closesocket()
}
```

- AcceptThread 생성
- 서버에서 게임 상태 매 프레임 (1/60s) 업데이트
  - 각 플레이어 입력 처리
  - 플레이어 상태 (위치, 체력 및 승패판정)
  - 아이템 상태
  - 총알 상태
  - 충돌 처리
- 매 프레임 모든 클라이언트에 게임 상태 send()

(수정)로우 레벨 디자인 – 스레드 및 동기화

서버 - Accept 스레드

```
AcceptThread
{
    while(g_running){
        accept(listensock)

        EnterCriticalSection()
        // 플레이어 초기화
        LeaveCriticalSection()

        EnterCriticalSection()
        // 플레이어 ID 부여
        LeaveCriticalSection()

        CreateThread(ProcessClient)
    }
    closesocket(clientsock)
}
```

- 클라이언트 접속 accept()
- 플레이어 초기 상태 초기화
- 접속한 클라이언트에 ID 부여
- 클라이언트 소켓 정보를 서버구조체에 전달
- ProcessClient 스레드 생성

서버 - ProcessClient 스레드

```
AcceptThread
{
    // 인자 ( 소켓, ID ) 받기
    // 클라이언트 정보 출력
    // 클라이언트 접속 종료 처리

    recv(clientsock, (char*) act, sizeof(act), 0)

    EnterCriticalSection()
    // 데이터 수신 후 ActionQueue에 삽입
    LeaveCriticalSection()

    return 0;
}
```

- 클라이언트의 입력 정보 수신
- 클라이언트 접속상태 관리
- 입력 정보 수신 후 액션 큐에 데이터 삽입

(수정)로우 레벨 디자인 – 스레드 및 동기화

클라이언트 - main 스레드 : 윈도우 프로시저

```
Main (WP_FINAL)
{
    CPlayLevel::UpdateLevel()
    CPlayLevel::DrawLevel()
}
```

클라이언트 - ClientThread 스레드

```
ClientThread
{
    // 소켓 생성
    Setsockopt(NODELAY)
    Connect()

    // 자신의 ID 수신
    Recv()

    // 서버 업데이트 주기에 맞추어 클라이언트 상태 업데이트
    Recv()

    EnterCriticalSection(&pThis->m_cs);
    EnterCriticalSection(&pThis->m_cs);
    pThis->m_recvQueue.push(recvData);
    LeaveCriticalSection(&pThis->m_cs);
}
```

프로토콜 및 패킷

- 패킷 정의 방식
  본 프로젝트는 TCP/IP 스트림 환경에서 데이터의 경계를 명확히 구분하고,
  std::vector와 같이 유동적인 데이터 크기를 효율적으로 처리하기 위해
  '가변 길이 패킷(Variable-Length Packet)' 방식을 채택했다.

  이는 고정 길이 헤더(Fixed-Length Header)와
  가변 길이 데이터(Variable-Length Data)를 조합하여 구현한다.

- 패킷의 기본 구조

1. 공통 헤더 (고정 길이 영역)

```
struct PacketHeader {
    short packetLength; // 패킷의 총 길이 (헤더 포함)
    short packetType;   // 패킷의 종류
};
```

2. 패킷 타입을 구분하기 위한 열거형

```
enum class PacketType : short {
    // --- 로비 레벨 ---
    C_SELECT_MAP = 101,      // [C→S] 클라가 맵 선택 ("MapType")
    S_LOBBY_INFO = 102,      // [S→C] 서버가 로비 정보 응답 ("LobbyInfo")
    C_READY_TO_PLAY = 103,   // [C→S] 클라가 게임 시작 준비 완료 ("ClientInfo" / "Ready")

    // --- 플레이 레벨 ---
    S_GAME_START = 201,      // [S→C] 서버가 양쪽 클라에게 게임 시작 명령 ("GameStart")
    CS_PLAYER_UPDATE = 301, // [C→S→C] 클라 상태 업데이트 (서버가 릴레이)
    S_GAME_OVER = 401        // [S→C] 서버가 게임 종료를 알림
};
```

3. 실제 데이터 예시

```
struct S_PLAY_UPDATE_Packet : public PacketHeader {
    // --- 고정 데이터 ---
    PlayerStatus status;        // 상대방 상태
    int          bulletNum;     // 총알 개수
    int          itemBoxNum;    // 아이템 박스 개수

    // --- 가변 데이터 (이 구조체 바로 뒤에 붙을 데이터) ---
    Bullet        bullets[bulletNum];
    ItemBox       itemBoxes[itemBoxNum];
};
```

역할분담

- 이도익

- 클라이언트 스레드 생성
- 클라이언트 스레드 – 에코 서버 테스트 (접속테스트)
- 서버 – 업데이트 (플레이어 위치 업데이트)
- 서버 – 업데이트 (총알 – 아이템 업데이트)

- 장윤서

- 비트맵 구조 수정
- 키 입력 버그 수정
- 서버에서 업데이트 한 구조체 클라이언트로 전송 구조 작성
- 서버 – 업데이트 (충돌 처리 (게임판정 포함))

- 조성욱

- 불릿을 플레이어에서 분리
- 일시정지 모드 제거
- 내 플레이어와 상대 플레이어 분리
- 그리기 구조 수정
- 업데이트 구조 수정
- 클라이언트 입력을 서버로 전송 – 서버 수신 (메인 스레드)

개발 일정 – 이도익

0주차 25.10.20 ~ 25.10.26

날짜	10 / 20	10 / 21	10 / 22	10 / 23	10 / 24	10 / 25	10 / 26
작업 내용			게임 내 총알 / 플레이어-총알 충돌 분리 (로직 변경)			텀프로젝트 추진 계획서 작성	

1주차 25.10.27 ~ 25.11.02

날짜	10 / 27	10 / 28	10 / 29	10 / 30	10 / 31	11 / 01	11 / 02
작업 내용	텀프로젝트 추진 계획서 작성	텀프로젝트 추진 계획서_v2 작성					서버 - 클라이언트 기본 구조 작성 send() - recv()

2주차 25.11.03 ~ 25.11.09

날짜	11 / 03	11 / 04	11 / 05	11 / 06	11 / 07	11 / 08	11 / 09
작업 내용	클라이언트 스레드 생성 후 에코서버 테스트	(수정) 소켓 함수 오류 코드 / 크리티컬 섹션 정의 , 클라이언트 스레드 생성 후 입력 정보 송/수신 로직 작성 및 테스트				~~클라이언트에서 서버로 보낼 구조체 작성 구조체 송수신 확인 테스트~~	~~단일 클라이언트 환경에서 struct PlayerStatus 작성 후 송수신 테스트~~

3주차 25.11.10 ~ 25.11.16

날짜	11 / 10	11 / 11	11 / 12	11 / 13	11 / 14	11 / 15	11 / 16
작업 내용	~~UpdatePlayer() 구조 설계~~	~~UpdatePlayers() 작성 후 클라이언트 테스트~~ (수정) 서버 – 클라이언트 임시 이동로직 작성 UpdatePlayer() 초안	(수정)임시 UpdatePlayer()작성 후 테스트 및 수정 완료	(수정) Cplayer 이동 로직 추가, 이동 변수 초기화	~~멀티 스레드 환경~~ ~~UpdatePlayers() 작성 후 클라이언트 테스트~~		

4주차 25.11.17 ~ 25.11.23

날짜	11 / 17	11 / 18	11 / 19	11 / 20	11 / 21	11 / 22	11 / 23
작업 내용	~~UpdateBullet() 구조 설계~~	~~UpdateBullet() 작성 후 클라이언트 테스트~~ (수정) 아이템, 총알 구조체 수정 서버 클라이언트 통신 구조체 변경작업	(수정) 총알 관련 함수 UpdateBullet() 초안 작성		~~UpdateItem() 구조 설계~~ (수정) 플레이어 낙사, 리스폰 로직 추가 및 테스트	~~UpdateItem() 작성 후 클라이언트 테스트~~ (수정) 아이템 랜덤 생성 및 낙하 로직 추가	

5주차 25.11.24 ~ 25.11.30

날짜	11 / 24	11 / 25	11 / 26	11 / 27	11 / 28	11 / 29	11 / 30
작업 내용	현재 까지의 진행 상황을 바탕으로 통합 작업 진행합니다.						

6주차 25.12.01 ~ 25.12.07

날짜	12 / 01	12 / 02	12 / 03	12 / 04	12 / 05	12 / 06	12 / 07
작업 내용	예외사항 테스트 및 디버깅						

개발 일정 – 장윤서

0주차 25.10.20 ~ 25.10.26

날짜	10 / 20	10 / 21	10 / 22	10 / 23	10 / 24	10 / 25	10 / 26
작업 내용	리펙토링 전역변수 및 전역 함수 정리	리펙토링 CBase (래퍼런스 카운트 관리) 클래스 작성	리펙토링 CObject (가상 부모 객체) 클래스 작성	리펙토링 비트맵매니저 작성	리펙토링 키매니저작성	텀프로젝트 추진 계획서 작성	

1주차 25.10.27 ~ 25.11.02

날짜	10 / 27	10 / 28	10 / 29	10 / 30	10 / 31	11 / 01	11 / 02
작업 내용	텀프로젝트 추진 계획서 작성	텀프로젝트 추진 계획서_v2 작성			리펙토링 기존 구조를 CBmpMgr를 사용하도록 변경	리펙토링 기존 구조를 CKeyMgr를 사용하도록 변경, 기존 버그 수정	서버 사용 변수 정의 작성

2주차 25.11.03 ~ 25.11.09

날짜	11 / 03	11 / 04	11 / 05	11 / 06	11 / 07	11 / 08	11 / 09
작업 내용	서버 사용 구조체 정의 작성			(수정) 메인스레드 – accept 스레드 분리	~~메인쓰레드 fps구조 설계~~	~~메인쓰레드 fps구조 작성 및 테스트~~	~~메인쓰레드 Update() 뼈대 작성~~

3주차 25.11.10 ~ 25.11.16

날짜	11 / 10	11 / 11	11 / 12	11 / 13	11 / 14	11 / 15	11 / 16
작업 내용	~~메인쓰레드 send() 작성~~	(수정) 서버 메인 스레드 업데이트 fps구조 설계 및 작성 / 액션 큐 생성, 게임월드 업데이트 기본 구조 작성		(수정) 게임 내 객체 충돌 처리 함수 Collision() 작성 및 캐릭터 이동 구현 완료 UpdatePlayer()	메인쓰레드 send() 테스트	메인쓰레드 맵 충돌 구조 설계	메인쓰레드 맵 충돌 구조 작성 및 테스트

4주차 25.11.17 ~ 25.11.23

날짜	11 / 17	11 / 18	11 / 19	11 / 20	11 / 21	11 / 22	11 / 23
작업 내용	메인쓰레드 맵 충돌 시 처리			(수정) 총알 – 플레이어, 아이템 – 플레이어 충돌 처리 완료	메인쓰레드 총알 충돌 구조 설계	메인쓰레드 총알 충돌 구조 작성 및 테스트	메인쓰레드 총알 충돌 시 처리

5주차 25.11.24 ~ 25.11.30

날짜	11 / 24	11 / 25	11 / 26	11 / 27	11 / 28	11 / 29	11 / 30
작업 내용	메인쓰레드 아이템 박스 충돌 구조 설계			(수정) 충돌처리 / 시작 조건 작성 및 UpdateCamera() 로직 변경	메인쓰레드 아이템 박스 충돌 구조 작성 및 테스트	메인쓰레드 아이템 박스 충돌 시 처리	플레이어가 중도 이탈 시 처리

6주차 25.12.01 ~ 25.12.07

날짜	12 / 01	12 / 02	12 / 03	12 / 04	12 / 05	12 / 06	12 / 07
작업 내용	비정상 종료 예외 처리	~~이후 일정은 예외상황 처리와 추가 리펙토링, 추가 구현사항을 구현합니다. 리펙토링 : CSoundMgr::LoadSound() 작성, CSoundMgr::PlaySound() 작성, CSoundMgr::StopSound() 작성, CSoundMgr::VolumeChange() 작성, 기존 구조를 CSoundMgr를 사용하도록 변경 추가 구현 : 클라이언트에 ReturnToMain() 작성, 게임방 스레드 설계, 여러 플레이어가 접속해서 여러 게임방에서 게임을 할 수 있게 구현, GameRoomTread 뼈대 작성, GameRoomThread를 담는 자료구조 선택 및 작성, CreateGameRoom() 작성, FindGameRoom() 작성, EnterGameRoom() 작성, 클라이언트에 JoinGame() 작성, GameStart() 작성, 대기 / 게임 상태 분기 상대를 너무 오래 찾을 때 예외 처리 동시 여러 방 생성 등 다양한 상황 테스트 방 최대 개수 제한 처리, 게임룸을 풀링으로 관리하게 구현~~					

개발 일정 – 조성욱

0주차 25.10.20 ~ 25.10.26

날짜	10 / 20	10 / 21	10 / 22	10 / 23	10 / 24	10 / 25	10 / 26
작업 내용			CPlayer 리팩토링	LevelManager 생성 MainLevel 생성	PlayLevel 생성	텀프로젝트 추진 계획서 작성	

1주차 25.10.27 ~ 25.11.02

날짜	10 / 27	10 / 28	10 / 29	10 / 30	10 / 31	11 / 01	11 / 02
작업 내용	텀프로젝트 추진 계획서 작성	텀프로젝트 추진 계획서_v2 작성		클라이언트 초기화 - 서버 연결 준비 LevelManager(Framework) 남기고 Player::update() 제외 모든 객체의 update()제거 목표: 내 플레이어만 움직이는 게임 만들기 (수정) 서버/클라 accept 테스트 추가 – Start 누르면 서버 연결			

2주차 25.11.03 ~ 25.11.09

날짜	11 / 03	11 / 04	11 / 05	11 / 06	11 / 07	11 / 08	11 / 09
작업 내용		~~클라이언트: 서버 연결 구현~~ ~~struct ServerPlay~~ ~~struct ClientPlay~~ (수정) ACTION 추가 및 통신 테스트 (Send – Recv)			~~클라이언트: 서버 연결 구현~~ ~~CMainLevel::Play() { 이후 socket(), connect() }~~ ~~recv() - struct ServerPlay~~ (수정) 이동 방식 리팩토링 – 서버에서 받은 pInfo 구조체로 이동 테스트		~~서버에 입력 넘겨주기~~ ~~send() - struct ClientPlay (MY_INPUT)~~ ~~서버 테스트 - 서버에서 입력 큐 구현~~

3주차 25.11.10 ~ 25.11.16

날짜	11 / 10	11 / 11	11 / 12	11 / 13	11 / 14	11 / 15	11 / 16
작업 내용		~~서버에서 받아 오는 정보에 따라 모든 객체의 위치 업데이트 & 그리기~~ ~~CPlayer::update()~~ ~~CPlayer::draw()~~ (수정) 클라이언트의 통신 스레드 구현	(수정) 키 입력 구조체 bool 기반으로 수정 및 테스트 완료	(수정) 여러명의 플레이어 접속을 위한 ID 통신 및 클라이언트의 플레이어를 배열로 생성 3번째 플레이어 리소스 제작 및 적용 테스트 완료	~~CEnemy::update()~~ ~~CEnemy::draw()~~ (수정) looking 이 클라이언트의 입력에 바로 반응하도록 수정		~~CItem::update()~~ ~~CItem::draw()~~

4주차 25.11.17 ~ 25.11.23

날짜	11 / 17	11 / 18	11 / 19	11 / 20	11 / 21	11 / 22	11 / 23
작업 내용		CBullet::update() CBullet::draw() (수정) CItem::update() CItem::draw() 아이템 그리기 성공, 총알 그릴 준비 (Info 구조체 기반)		(수정) 총알이 한쪽 방향으로만 나가는 문제 수정, 플레이어의 정보창 그리기 (연결여부에 따라), 총알 궤적 (그라데이션) 추가 및 사운드 재생	~~eGameState에 따른 목숨조정 게임 시스템 판정 출력~~		~~모든 플레이어의 INFO UI 출력~~ (수정) 서버와 연결이 끊어진 경우 강제 Game Over 처리 (패배)

5주차 25.11.24 ~ 25.11.30

날짜	11 / 24	11 / 25	11 / 26	11 / 27	11 / 28	11 / 29	11 / 30
작업 내용	현재 까지의 진행 상황을 바탕으로 통합 작업 진행합니다.						

6주차 25.12.01 ~ 25.12.07

날짜	12 / 01	12 / 02	12 / 03	12 / 04	12 / 05	12 / 06	12 / 07

작업 내용	예외사항 테스트 및 디버깅