I n t r o d u c e .

서버 프로그래머 김승환

김승환

KIM SEUNG HWAN

서버 프로그래머 포트폴리오

"**새로운 도전**과 **색다른 시도**를
즐길 줄 아는 게임 프로그래머를
목표로 하고 있습니다."

생일. 1999. 11. 15

전화번호. 010-8131-2128

E-메일. hpy469@gmail.com

주소지. 충청남도 당진시

최종학력.

2018.02 | 신평고등학교 졸업

2024.02 | 한국공학대학교(구 한국산업기술대학교) 게임공학과 졸업예정

Skills.

언어.
- C
- C++11
- Python
- Lua Script

서버.
- IOCP
- Multithread
- MS SQL

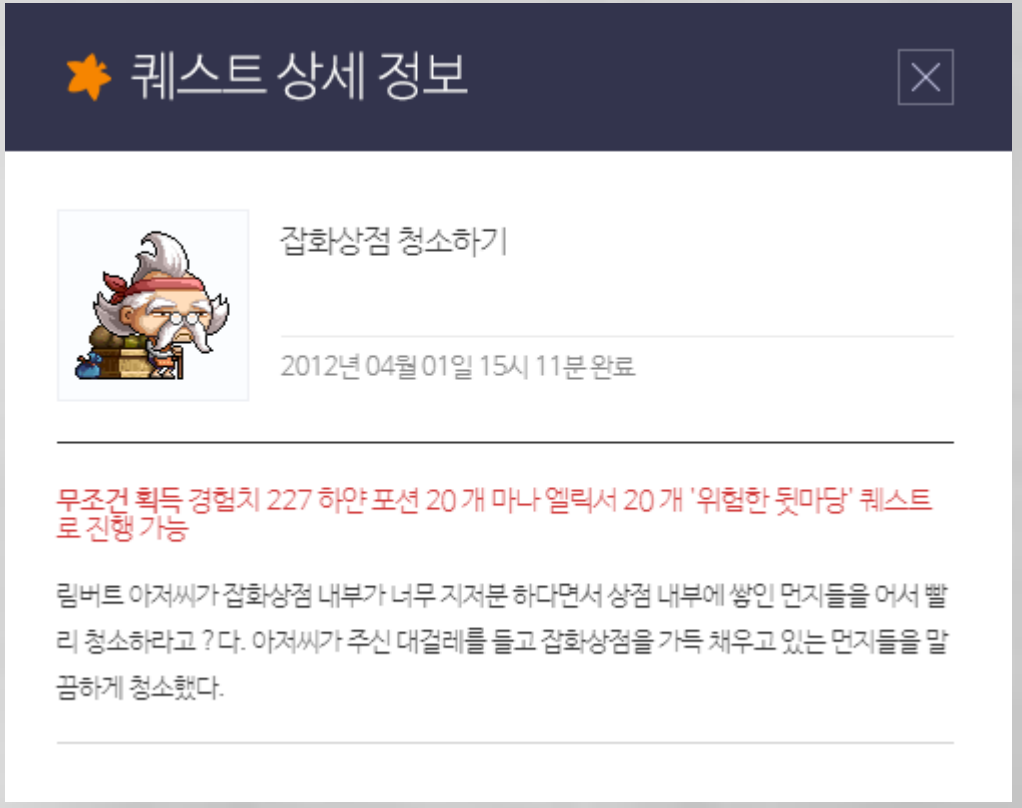
클라이언트.
- Windows API
- OpenGL
- Unity

ETC.
- Git

I n t r o d u c e .

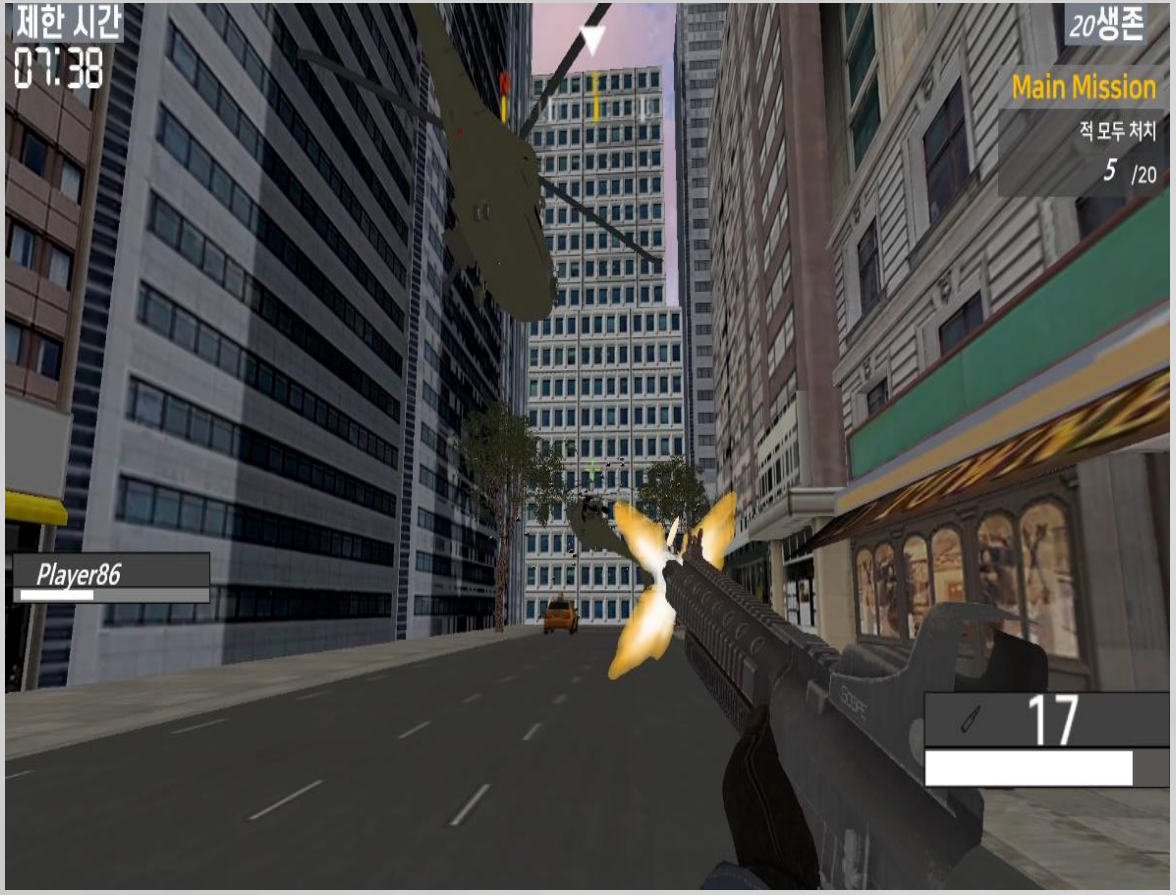
'메청년' 김승환

- 초등학생 시절 시그너스기사단이 출시했던 즈음 처음 시작 (해당 계정 삭제로 기록X)
- 남아있는 가장 오래된 기록 2012년 04월 01일
- 현재도 엘리시움 서버에서 메이플 라이프를 소소하게 즐기는 중



Contents.

01. 졸업작품



게임명	Revenger (리벤저)
게임 장르	3D 멀티플레이 FPS
개발 기간	2022.07 ~ 2023.07
개발 인원	3명 (클라이언트 1명, 서버 2명)
사용 기술	C++11, 17(Filesystem) 멀티스레드 IOCP 서버 이중화 DirectX12
역할 (구현 내용)	서버 프로그래머 - 플레이어 이동, 회전, 공격 등 각종 동기화 작업 - 컨텐츠 구현 (미션, 적 처치, 점령) - 로비 구현 (방 생성, 방 진입, 랜덤 매칭) - 서버 이중화를 통한 고가용성 구현
Git	https://github.com/kiption/XD
시연 영상	https://youtu.be/60aF2ufRuNE?si=LdHRa7Q258jG22Zy
특이사항	학과 우수 추천작으로 선정되어 2023년 한국공학대전 디지털전시관에 전시하였음.

01. 구현 내용_ 본인 역할

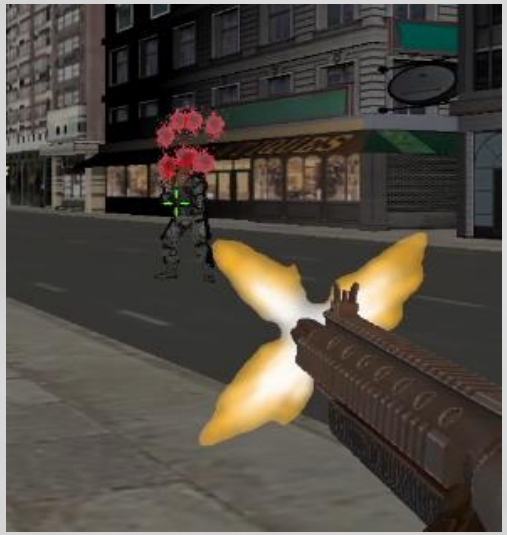
▶ 로비_ 방 생성, 방 진입, 랜덤 매칭

▶ 게임 방_ 역할 선택, 준비, 게임 진입

▶ 멀티 플레이_ 플레이어 동기화

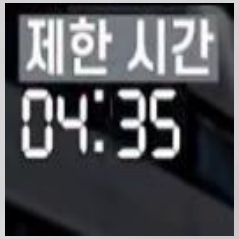
▶ 미션_ 적 처치, 점령

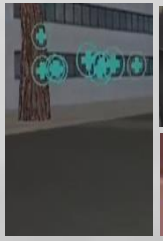
▶ 서버 내 충돌검사_ 공격/피격/맵 충돌 검사

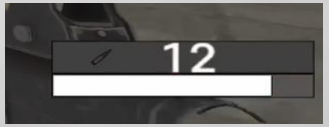


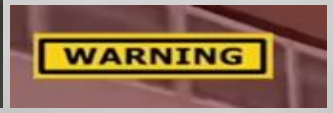
▶ 채팅

▶ 기타_ 시간, 회복, 총알 및 HP, 경고 등 서버작업

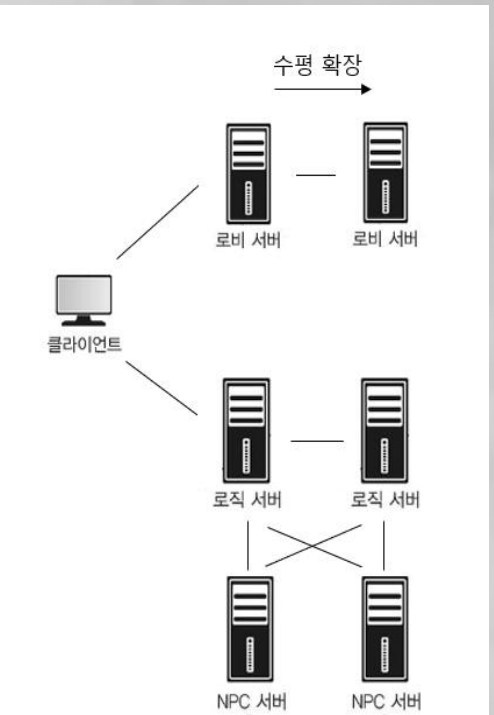








▶ 서버 이중화

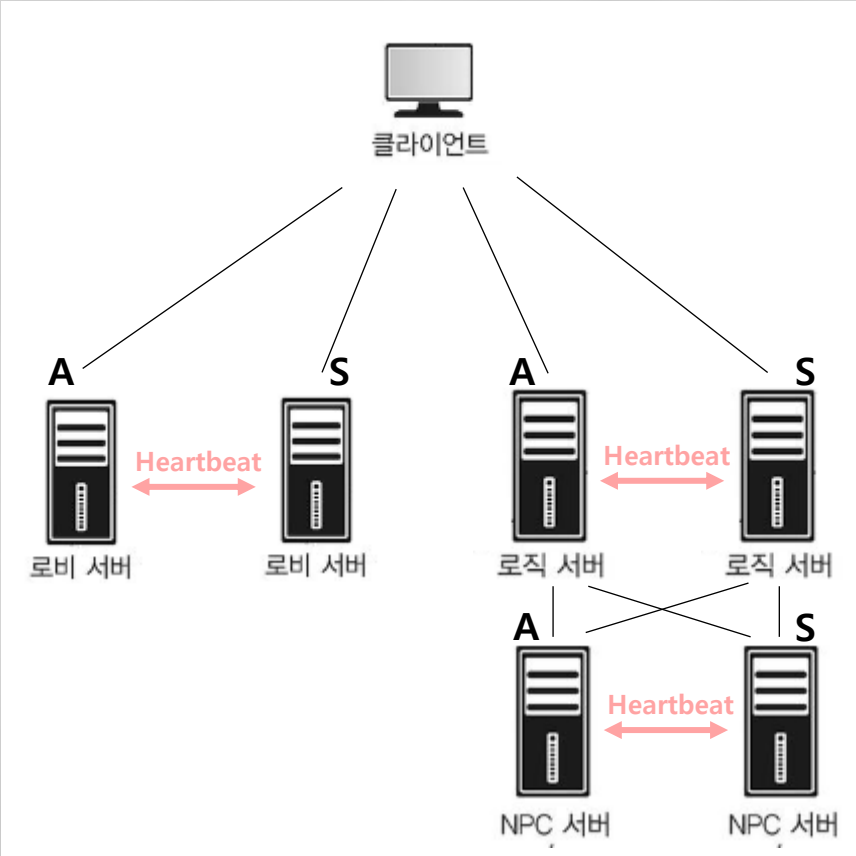


서버 프로그래머로서...

멀티플레이, 게임 컨텐츠, 로비 관련 여러 서버 작업들을 맡았고,

중점연구과제로 서버 이중화를 구현하였습니다.

01. 중점연구과제_ 서버 이중화



▶ 서버 다운은 어떻게 알아내는가?

: 서버들은 **주기적**으로 **Heartbeat**를 패킷을 통해 주고 받으면서
서로 살아있는지 감시하도록 구현하였습니다.

ex. 서버A와 서버B가 있을 때, 서버B로 부터 Heartbeat 패킷을 오랫동안 받지 못했다!
→ 서버A는 서버B가 다운되었다고 판단하고 후속 조치를 취합니다.

▶ 서버 다운을 감지하면 어떻게 되는가?

① 가동 중인 다른 서버는 다운된 서버와 연결을 끊고, 서버를 재실행합니다.
(이때 저는 ShellExecute를 사용하여 서버 exe파일을 실행하는 방법을 사용하였습니다.)

```
case SESSION_EXTENDED_SERVER:
{
    extended_servers[target_id].s_lock.lock();
    if (extended_servers[target_id].s_state == ST_FREE) {
        extended_servers[target_id].s_lock.unlock();
        return;
    }
    closesocket(extended_servers[target_id].socket);
    extended_servers[target_id].s_state = ST_FREE;
    extended_servers[target_id].s_lock.unlock();
    cout << "Server[" << extended_servers[target_id].id << "]의 다운이 감지되었습니다." << endl;

    // 서버 재실행
    wchar_t wchar_buf[10];
    wsprintfW(wchar_buf, L"%d", 10 + target_id);   // 십의자리: Standby 1, Active 2, 일의자리: 서버ID

    ShellExecute(NULL, L"open", L"Server.exe", wchar_buf, L".", SW_SHOW);
```

01. 중점연구과제_ 서버 이중화

▶ **서버 다운을 감지하면 어떻게 되는가? (계속)**

② 되살아난 서버와 재연결을 해야 하는데...

기존의 Connect를 사용하면 다른 서버와의 Connect가 성공할 때까지 Blocking되는 것이 문제가 되었습니다.

하지만 Active 서버는 다른 서버와 Connect를 시도하는 동안에도 게임 로직을 계속해서 수행해야 합니다.

이러한 이유로 수평 확장된 서버 간 연결은 Connect가 아닌 비동기 Connect를 사용하게 되었습니다.

그런데, 비동기 Connect는 기존의 Connect와 달리 특수한 방법을 사용해야 했습니다.

```
SOCKET temp_s = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
GUID guid = WSAID_CONNECTEX;
DWORD bytes = 0;
LPFN_CONNECTEX connectExFP;
::WSAIoctl(temp_s, SIO_GET_EXTENSION_FUNCTION_POINTER, &guid, sizeof(guid), &connectExFP, sizeof(connectExFP), &bytes, nullptr, nullptr);
closesocket(temp_s);
```

첫째로, **WSAIoctl( )**를 통해 별도의 함수 포인터를 얻는 작업이 필요했습니다.

WSAIoctl를 사용하기 위해서

1. 함수를 가져오기 위한 **임시 소켓**(temp_s)
2. ConnectEx API의 함수 포인터를 식별해주기 위한 WSAID_CONNECTEX가 저장된 **GUID**(guid)
3. 기존의 Connect( )처럼 사용하기 위한 **LPFN_CONNECTEX 함수 포인터**(connectExFP)

를 만들어주었습니다.

01. 중점연구과제_ 서버 이중화

▶ **서버 다운을 감지하면 어떻게 되는가? (계속)**

② (계속)

```
SOCKADDR_IN ha_server_addr;
ZeroMemory(&ha_server_addr, sizeof(ha_server_addr));
ha_server_addr.sin_family = AF_INET;
ex_server_sock = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
int ret = ::bind(ex_server_sock, reinterpret_cast<LPSOCKADDR>(&ha_server_addr), sizeof(ha_server_addr));
if (ret != 0) { cout << "Bind Error - " << ret << endl; cout << WSAGetLastError() << endl; exit(-1);  }
```

또한 실제 사용할 소켓(ex_server_sock)을 **bind** 하는 작업도 필요했습니다.

아래는 이렇게 가져온 ConnectEx 함수를 사용하여 실제 비동기 연결을 시도하는 코드입니다.

```
ZeroMemory(&ha_server_addr, sizeof(ha_server_addr));
ha_server_addr.sin_family = AF_INET;
ha_server_addr.sin_port = htons(target_portnum);
inet_pton(AF_INET, IPADDR_SERVER1, &ha_server_addr.sin_addr);

BOOL bret = connectExFP(right_ex_server_sock, reinterpret_cast<sockaddr*>(&ha_server_addr), sizeof(SOCKADDR_IN), nullptr, 0, nullptr, &con_over->overlapped);
if (FALSE == bret) {
    int err_no = GetLastError();
    if (ERROR_IO_PENDING == err_no)
        cout << "Server Connect 재시도 중...₩n" << endl;
    else {
        cout << "ConnectEX Error - " << err_no << endl;
        cout << WSAGetLastError() << endl;
```

01. 중점연구과제_ 서버 이중화

▶ **서버 다운을 감지하면 어떻게 되는가? (계속)**

③ 이후의 복구 방법은 다운된 서버가 **'Standby였는지 Active였는지'**, **'로직(또는 로비) 서버였는지 NPC서버였는지'**에 따라 달라집니다.

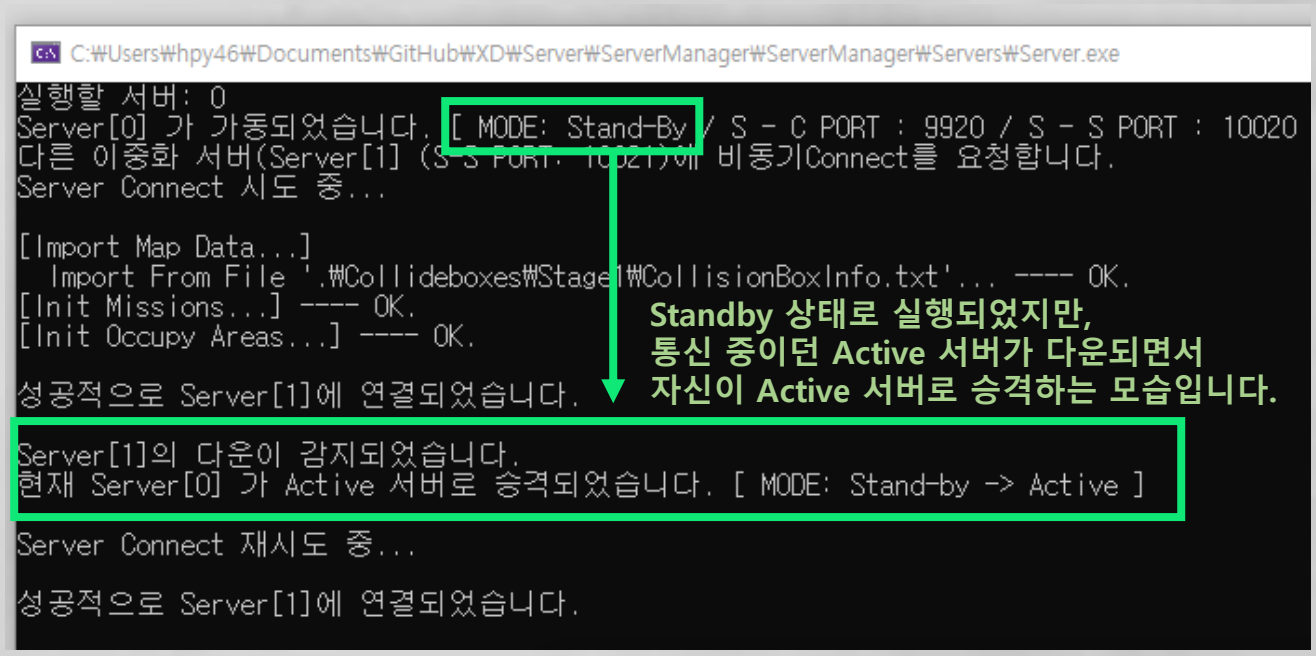
1) 다운된 서버가 로직 또는 로비 서버

① Standby 상태

: 서버가 되살아나고 연결도 잘 되었다면, 더 이상의 작업은 없습니다.

② Active 상태

: Failover가 이루어집니다. 가동중인 다른 대기서버가
Active 상태로 승격되어 서비스를 이어받아 하게 됩니다.
이때 새로 Active가 된 서버는 접속 중인 모든 클라이언트에게
Failover가 이루어졌음을 알리는 패킷을 보내
지금부터 네트워크 통신을 자신과 하도록 합니다.



2) 다운된 서버가 NPC 서버

① Standby 상태의 NPC서버

: 1) - ① 과 동일합니다.

② Active 상태의 NPC서버

: 1) - ② 와 비슷하지만, 클라이언트와 직접 연결되지 않았기 때문에 클라이언트가 아닌 로직 서버에 Failover가 이루어졌음을 알립니다.

01. 중점연구과제_ 서버 이중화

▶ **데이터 복제 (Data Replica)**

: Active 상태인 서버만 클라이언트와 통신을 하도록 하였습니다. (NPC서버(Active)의 경우 클라이언트가 아닌 로직 서버와 통신)

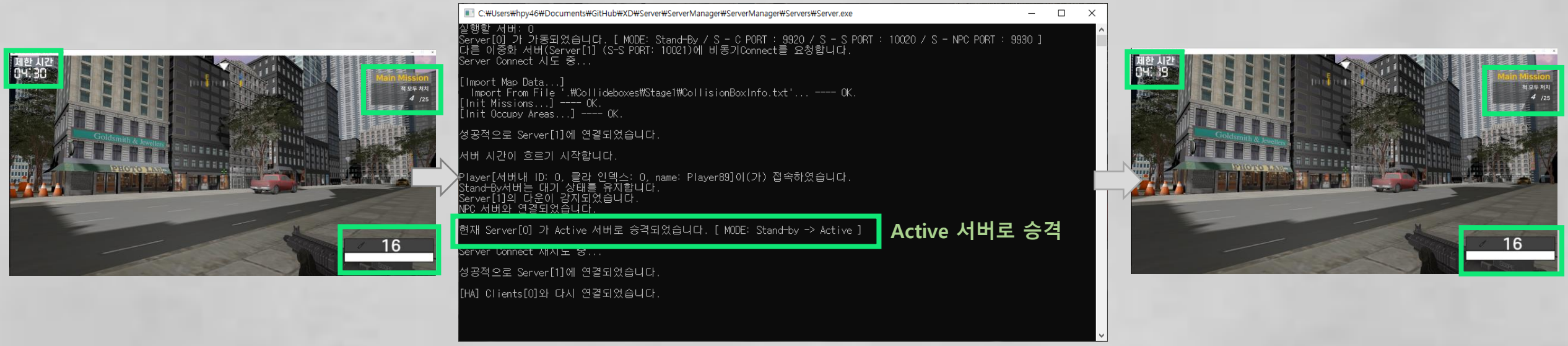
그렇기 때문에 Standby 서버는 인게임 정보가 업데이트되지 않아 Failover 이후 서비스를 이어받게 되었을 때

세션 정보는 물론이고 여러 인게임 정보들이 초기화되는 문제가 생깁니다.

이러한 Failover 이후 정보가 초기화되는 문제를 막기 위해 수평 확장된 서버간 데이터를 복제할 필요가 생겼습니다.

Active 서버는 주기적으로 Standby 서버에게 자신이 갖고 있는 인게임 정보들(ex. 세션 정보, 미션 상태 등)을 패킷을 통해 보내주도록 하였고,

Standby 서버는 이를 받아 자신의 세션 및 여러 인게임 객체들의 정보를 패킷의 정보들로 업데이트하도록 구현하였습니다.



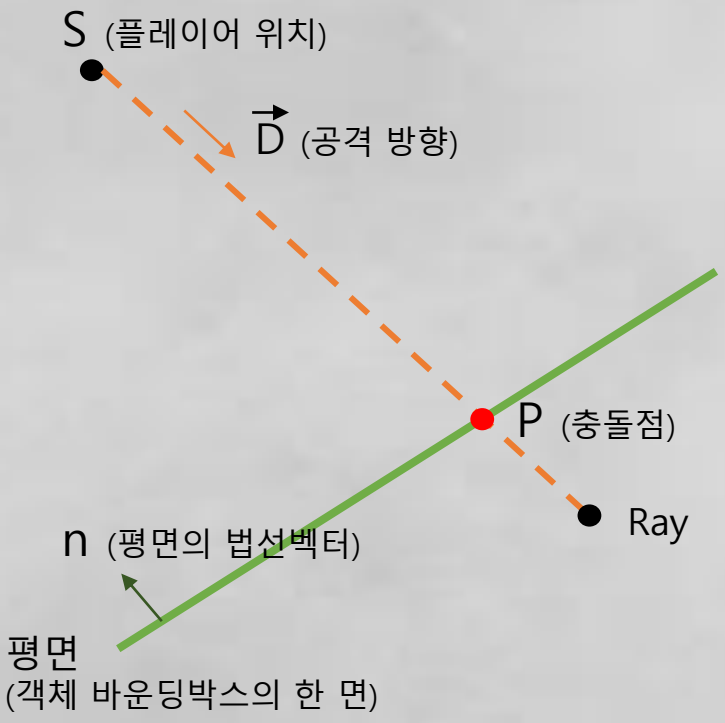
▲ 클라이언트가 연결되어 있는 상태에서 Active 서버를 강제 종료하여 Failover가 이뤄진 후에도

클라이언트 정보(위치, HP, 총알, 미션 등)가 그대로 유지되며, 서버 시간도 잘 흘러가는 모습입니다.

01. 기타 구현 사항

▶ **Ray Cast 구현**

유클리드 벡터 공간에서 광선-평면의 교점 구하는 방식을 이용해 Ray Cast를 직접 구현하여
플레이어와 NPC의 사격 공격에 대한 충돌 처리에 사용하였습니다.



충돌점(P)은 어떻게 구하는가?

1. 공간 상의 한 점 S (플레이어 위치)에서
   D 방향 (플레이어 카메라 방향) 으로 뻗어 나가는
   광선의 방정식을 구합니다.

2. 충돌 검사하려는 객체 바운딩 박스의 한 면에 대한
   평면의 방정식을 구합니다.

3. 광선의 방정식과 평면의 방정식이 교점 P (충돌점)을 구합니다.

4. 점 P가 평면 내부에 존재하는지 검사합니다.
   (점 P의 x, y, z 값을 갖고 사각형의 좌측상단 점과 우측하단 점 사이에 있는 지 확인합니다.)

5. 2~4의 과정을 객체 바운딩 박스의 면들 중 은면이 아닌 면에 대해서 진행합니다.
   (은면인지에 대한 여부는 평면 법선벡터와 벡터 D를 내적하여 검사합니다.)

6. 지금까지의 결과로 나온 충돌점들 중 S와의 거리가 가장 가까운 점을 구하고,
   이를 Ray Cast의 최종 결과로 반환합니다.

01. 기타 구현 사항

▶ Ray Cast 구현 (계속)

```
// Ray Cast
XMFLOAT3 MyRaycast (XMFLOAT3 start, XMFLOAT3 direction, float range_limit, XMFLOAT3 box_center, float box_width, float box_height, float box_length) {
    if (GetDistance(start, box_center) > range_limit) return NONCOLLIDE;

    XMFLOAT3 p[8];
    for (int i = 0; i < 8; ++i) {
        p[i] = GetBBPoints(box_center, box_width, box_height, box_length, i);
    }

    XMFLOAT3 tmp_results[6];
    tmp_results[0] = IntersectCheck (start, direction, range_limit, p[0], p[1], p[2]);
    if (tmp_results[0] != NONCOLLIDE) {
        if (!Check_PointInPlane(tmp_results[0], p[0], p[3])) tmp_results[0] = NONCOLLIDE;
    }
    tmp_results[1] = IntersectCheck(start, direction, range_limit, p[1], p[0], p[5]);
    if (tmp_results[1] != NONCOLLIDE) {
        if (!Check_PointInPlane(tmp_results[1], p[1], p[4])) tmp_results[1] = NONCOLLIDE;
    }
    ...

    XMFLOAT3 final_result = NONCOLLIDE;
    float min_dist= FLT_MAX;
    for (int i = 0; i < 6; ++i) {
        if (tmp_results[i] == NONCOLLIDE) continue;
        float cur_dist = GetDistance(start, tmp_results[i]);
        if (cur_dist < min_dist) {
            min_dist = cur_dist;
            final_result = tmp_results[i];
        }
    }

    return final_result;
}
```

바운딩박스의 점 8개를 구해서 p[i]에 저장합니다.

광선과 평면의 교점을 구하는 함수입니다.
(다음 페이지에 코드를 첨부하였습니다.)

여기서 3~5번째 파라미터는
사각형(평면)을 만들기 위한 세 점입니다.

ex. P[0] P[1]

P[2]

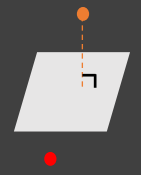
위 과정을 tmp_results[2]부터 tmp_results[5]까지 동일하게 반복합니다.

위 과정에서 나온 교점들 중
플레이어 위치(start)와
가장 가까운 한 점을
선택하는 과정입니다.

IntersectCheck( )의 결과로 나온 교점이
평면 안에 있는지 검사하는 함수입니다.

ex.

교점의 x, y, z 값 모두
평면 안에 있는 경우
→ **return true**



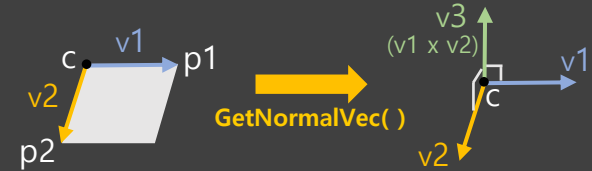
교점의 x, y, z 값 중
단 하나라도 평면 밖에
존재하는 경우
→ **return false**

01. 기타 구현 사항

▶ Ray Cast 구현 (계속)

```
// Intersect Check : 광선과 평면의 교점을 구합니다.
XMFLOAT3 IntersectCheck(XMFLOAT3 start, XMFLOAT3 direction, float range_limit,₩
      XMFLOAT3 plane_center, XMFLOAT3 plane_p1, XMFLOAT3 plane_p2)
{
    XMFLOAT3 normal = GetNormalVec(plane_center, plane_p1, plane_p2);
    // 광선과 평면이 평행 -> 충돌X
    if (DotProduct(direction, normal) == 0) return NONCOLLIDE;

    XMFLOAT3 v{ 0,0,0 };
    bool infinite_ray = false;
    if (range_limit >= FLT_MAX) {
        v = direction;
        infinite_ray = true;
    }
    else {
        XMFLOAT3 end = Add(start, MultiplyScalar(direction, range_limit));
        v = Substract(end, start);
        infinite_ray = false;
    }

    float t = Get_ParamT_RayDescription(start, direction, plane_center, plane_p1, plane_p2);

    // 시작점이 평면 뒤에 있음. -> 광선을 쏘는 방향 반대에 평면이 있음
    if (t < 0) return NONCOLLIDE;

    // 끝점이 평면 앞에 있음. -> 충돌점이 Ray의 사거리 끝보다 멀리 있는 것.
    if ((!infinite_ray) && (t > 0)) return NONCOLLIDE;

    return Add(start, MultiplyScalar(v, t));
}
```

세 점을 가지고 두 벡터를 만든 다음
이 두 벡터를 서로 외적하여
법선벡터를 구하는 함수입니다.



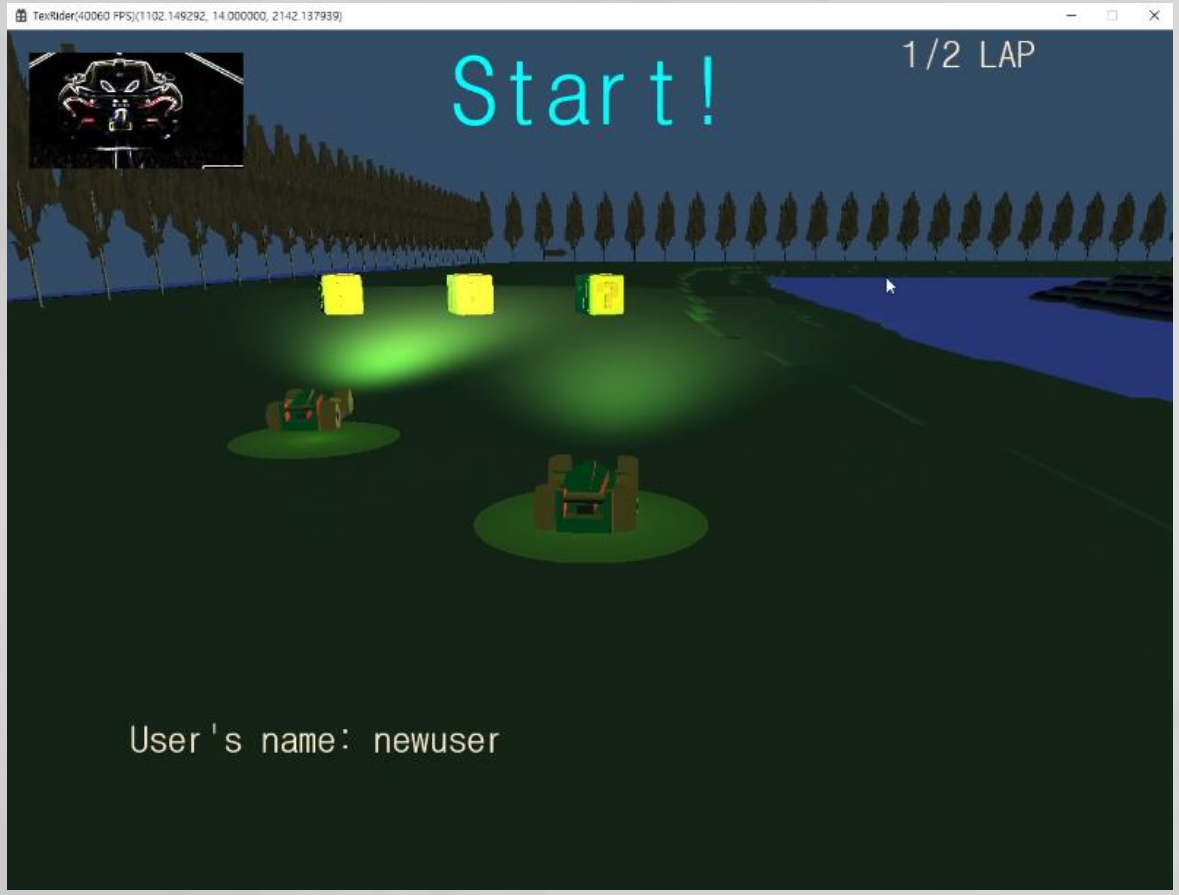
광선 방정식의 매개변수(t)를 구하는 함수입니다.

```
float Get_ParamT_RayDescription(XMFLOAT3 start, XMFLOAT3 direction, XMFLOAT3 plane_center,₩
    XMFLOAT3 plane_p1, XMFLOAT3 plane_p2)
{
    float d = Get_ValueD_PlaneDescription(plane_center, plane_p1, plane_p2);
    XMFLOAT3 normal = GetNormalVec(plane_center, plane_p1, plane_p2);
    return (-1.0f * (d + DotProduct(start, normal))) / DotProduct(direction, normal);
}
```

평면 방정식에서의 상수항(d)을 구하는 함수입니다.

```
float Get_ValueD_PlaneDescription(XMFLOAT3 plane_center, XMFLOAT3 plane_p1,₩
    XMFLOAT3 plane_p2)
{
    XMFLOAT3 normal = GetNormalVec(plane_center, plane_p1, plane_p2);
    float ax0 = normal.x * plane_center.x;
    float by0 = normal.y * plane_center.y;
    float cz0 = normal.z * plane_center.z;
    return -1.0f * (ax0 + by0 + cz0);
}
```

02. 네트워크 게임 프로그래밍



게임명	(없음)		
게임 장르	3D 멀티플레이 레이싱 게임		
개발 기간	2022.11 ~ 2022.12		
개발 인원	3명 (전 인원 서버작업에 참여)		
사용 기술	C++11 TCP/IP DirectX12		
역할 (구현 내용)	공통	로그인 서버	로직 서버
	- 클라이언트 연결 작업	- 계정등록 기능 구현	- 플레이어 이동 및 회전 - 멀티플레이를 위한 동기화 작업 - 플레이어 간 충돌검사 - 아이템(부스터) 구현 - 주행시간 측정 및 완주여부 검사
Git	https://github.com/kiption/Network_Game_Programming		
시연 영상	https://youtu.be/GooATFiHUyA		

02. 구현 내용_ 본인 역할

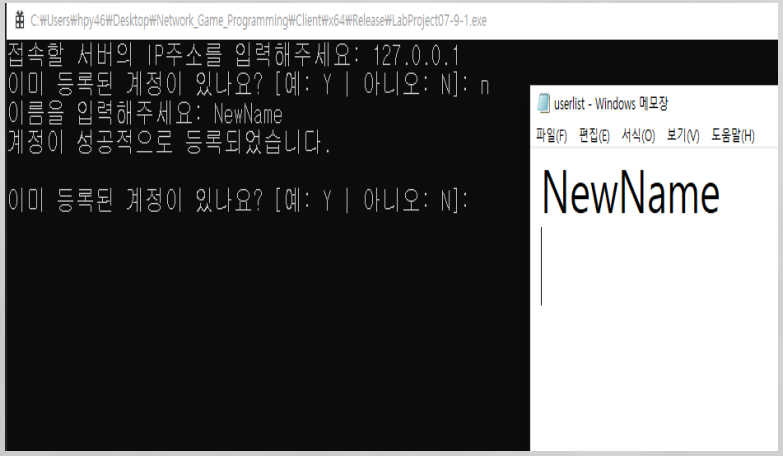
▶ 서버-클라이언트 연결

```
login_sock = socket(AF_INET, SOCK_STREAM, 0);
if (login_sock == INVALID_SOCKET)
    err_quit("socket()");

struct sockaddr_in login_server_addr;
memset(& login_server_addr, 0, sizeof(login_server_addr));
login_server_addr.sin_family = AF_INET;
login_server_addr.sin_addr.s_addr = inet_addr(SERVERIP);
login_server_addr.sin_port = htons(LOGIN_SERVER_PORT);
int retval = connect(login_sock, (struct sockaddr*)₩
    & login_server_addr, sizeof(login_server_addr));
if (retval == SOCKET_ERROR)
    err_quit("connect()");
```

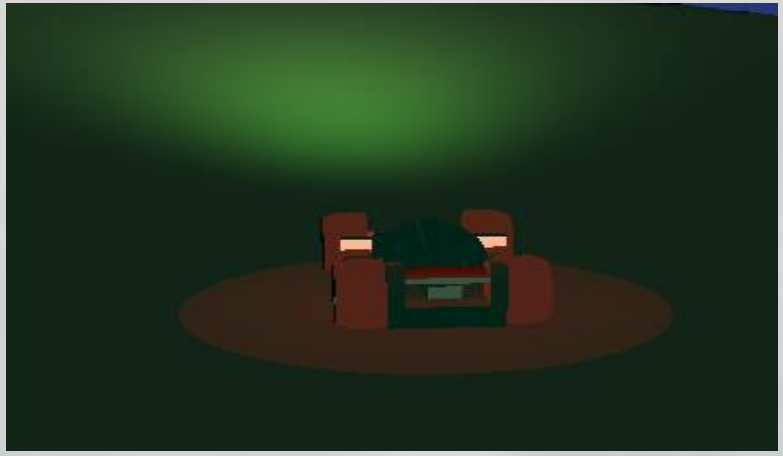
▶ 플레이어 이동/회전 및 멀티플레이 구현

▶ 플레이어 간 충돌검사

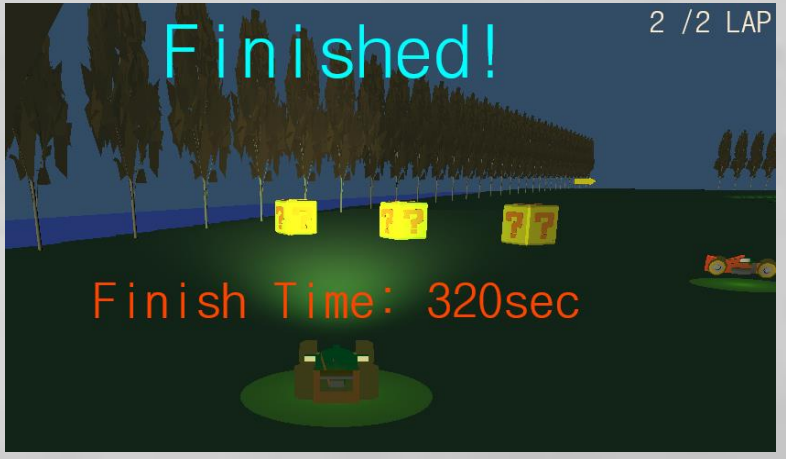
▶ 계정등록 기능 구현



▶ 아이템(부스터) 효과 구현

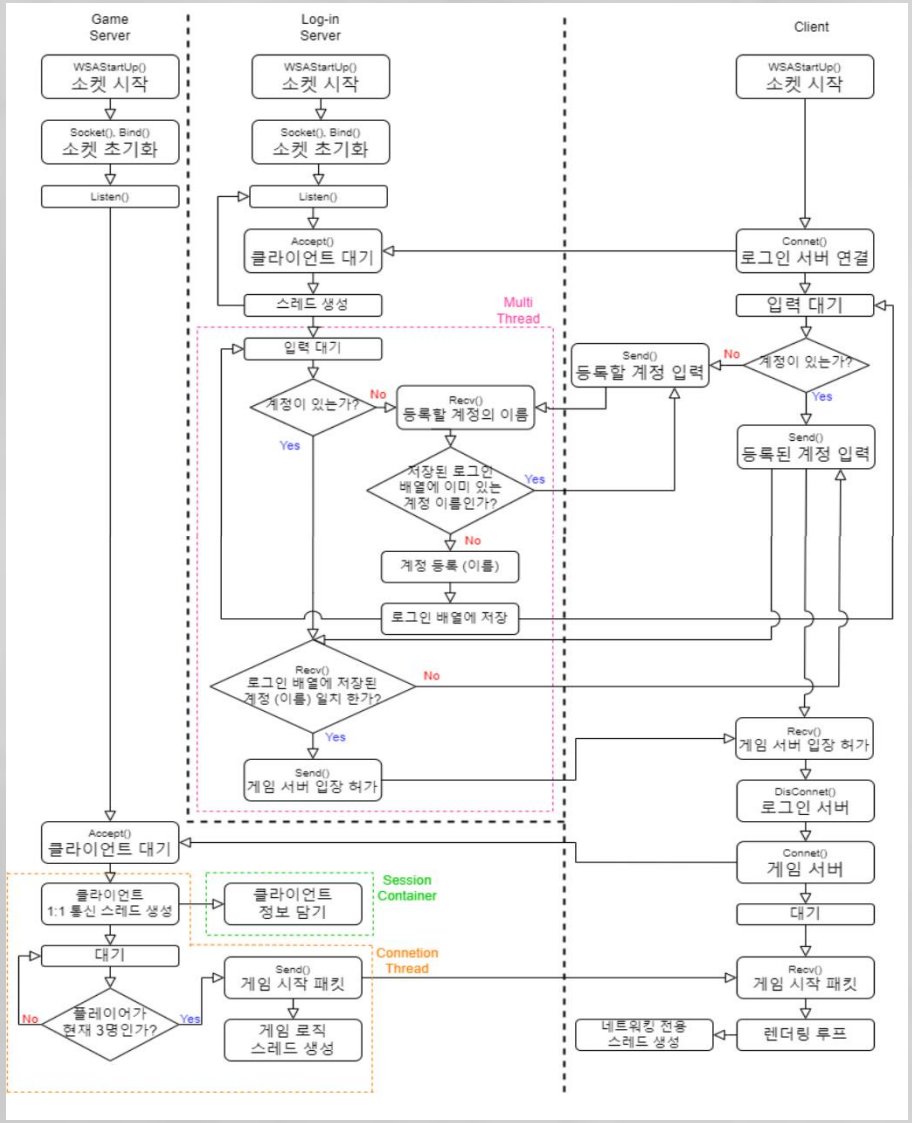


▶ 주행시간 측정 및 완주여부 검사

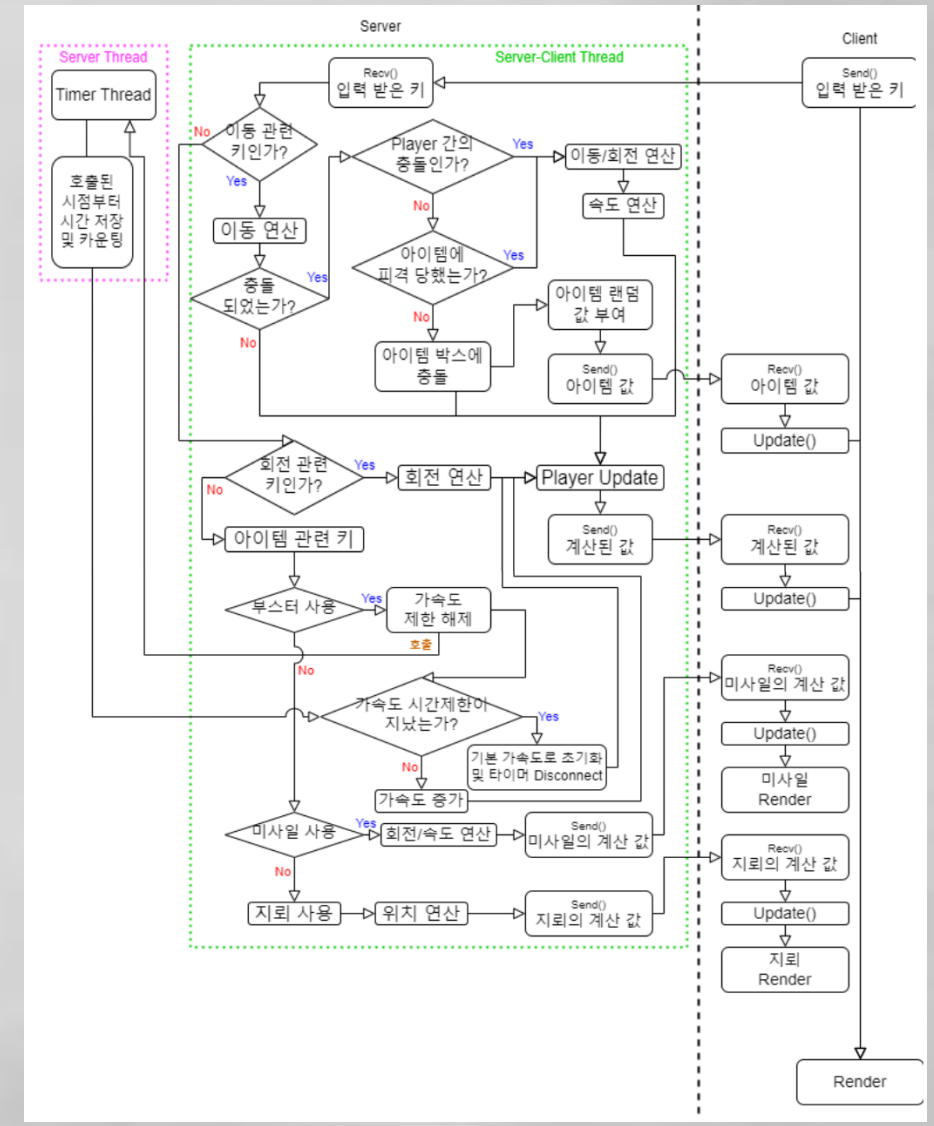


02. 플로우 차트

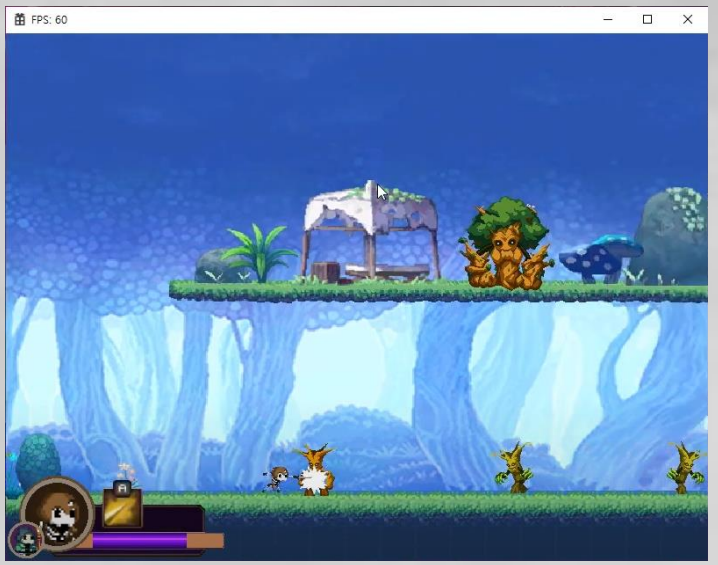
▶ Flow Chart_ Login Server



▶ Flow Chart_ Main Server



03. 기타 작업물



게임명	Skul: The Hero Slayer (모작)
게임 장르	2D 액션 플랫포머
개발 기간	2021.05 ~ 2021.06
개발 인원	2명 (본인 역할: 조작 및 스킬 구현, 캐릭터 교체 기능, UI 구현, 지형 충돌처리)
사용 기술	C, C++, Windows API

게임명	슈퍼 마리오 브라더스 (모작)
게임 장르	2D 액션 플랫포머
개발 기간	2021.10 ~ 2021.12
개발 인원	1명
사용 기술	Python, Pico-2D(SDL기반)

03. 기타 작업물

게임명	Galaxy Guardian
게임 장르	3D 싱글플레이 슈팅게임
개발 기간	2021.11 ~ 2021.12
개발 인원	2명 (본인 역할: 객체 움직임 구현, 충돌 처리)
사용 기술	C++, OpenGL

게임명	(없음)
게임 장르	3D 싱글플레이 TPS
개발 기간	2023.05 ~ 2023.06
개발 인원	1명
사용 기술	C#, Unity