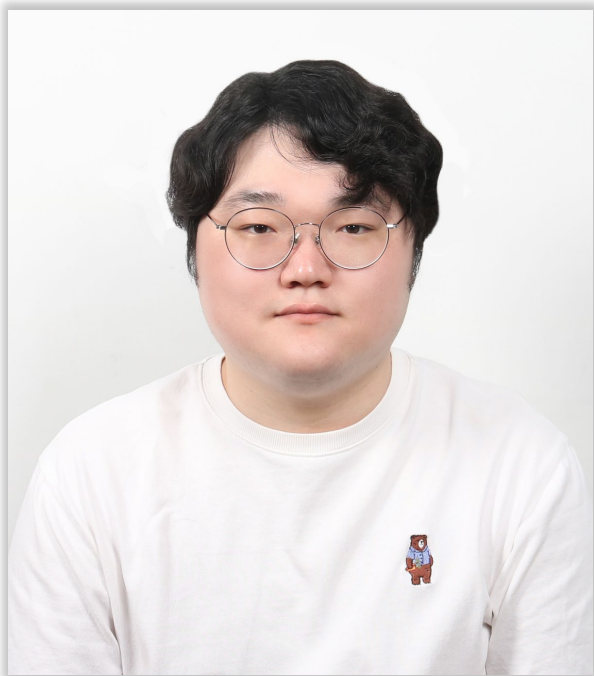


IM HAE IN  
Game Developer

# Portfolio



## 임해인 IM HAEIN

1998.03.03. 경기도 시흥시

Tel. 010-8991-9593

Email. devhaein0303@gmail.com

Git. <https://github.com/haein0303>

### ▪ 학력

2024.

한국 공학대학교 졸업 예정

주) 게임공학과  
부) 컴퓨터공학과

2016.

대진 디자인고등학교 졸업

컴퓨터 미디어 디자인과

### ▪ 이력

2024.01 ~ .

무브인터랙티브 개발본부 라이브팀

모바일 게임 서버 프로그래머

2023.10 ~ 12.

펼어비스 테크 인턴

게임 플레이 프로그래머 ( 신작 콘텐츠 개발팀 )

2022.11 ~ 2023.10.

컴세바아이티 탐구학원 은행점

프로그래밍 강사

### ▪ 기타

2017.06 ~ 2018.12.

STUDIO.BLUECAT::NERU

창업동아리 대표

## 이력 / 학력 / 활동

2017



한국공학대학교 2017.02. ~ 2024.02.  
- 게임공학과 / (부) 컴퓨터공학과



STUDIO.BLUECAT::NERU 2017.06 ~ 2018.12  
- 대표(창업동아리)  
- 게임 기획, PM, 운영



2019

공군 복무 2019.01. ~ 2021. 11.  
- 통신 하사 전역

2022



(주)360미디어 2022.07. ~ 2022.08.  
- 현장실습생  
- 독도 전시관 구성 사업 참여  
- 기획, PM, 클라이언트 프로그래머

2023



컴세바 아이티 탐구학원 2022.11 ~ 2023.10.  
- 프로그래밍 강사



펼어비스 2023.10. ~ 2023.12.  
- 테크 인턴쉽  
- 신작콘텐츠개발1팀  
- 게임플레이프로그래머  
- 붉은사막, 도깨비, Plan8



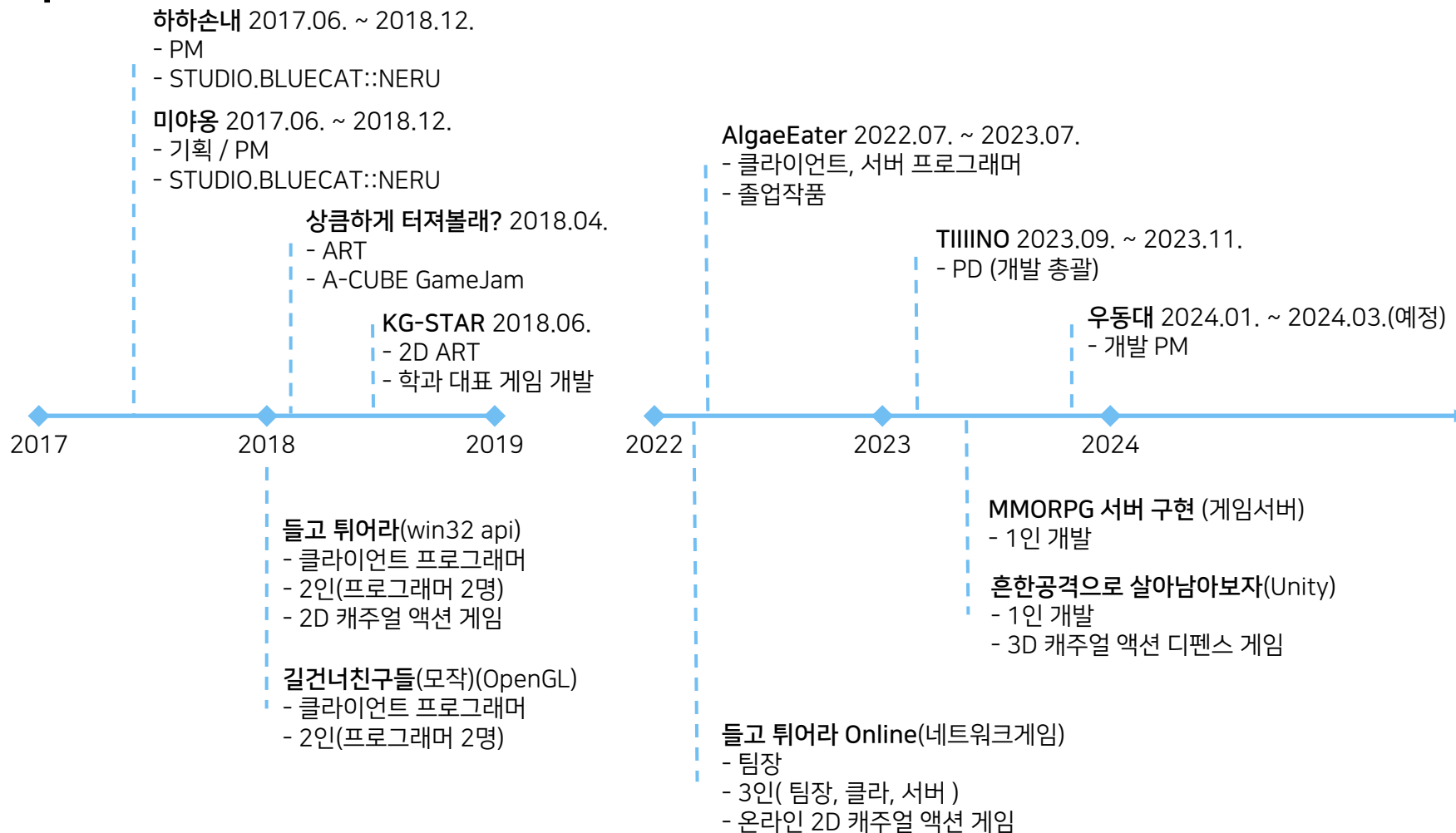
2024



무브인터랙티브 2024.01.  
- 모바일 게임 서버 프로그래머  
- 개발본부 라이브팀  
- 연희몽상:소녀대첩



## 프로젝트



## 학과 팀프로젝트



## AlgaeEater

알지이터

장르 : 3D 액션

개발기간 : 2022.07. ~ 2023.07.

사용도구 : C++17, IOCP, DirectX12

개발인원 : 3명 ( 클라이언트 2명, 서버 1명)

Git : <https://github.com/haein0303/AlgaeEater>

역할 :

클라이언트

- 멀티쓰레드 클라이언트
- 이동 보간
- 콘텐츠 구현



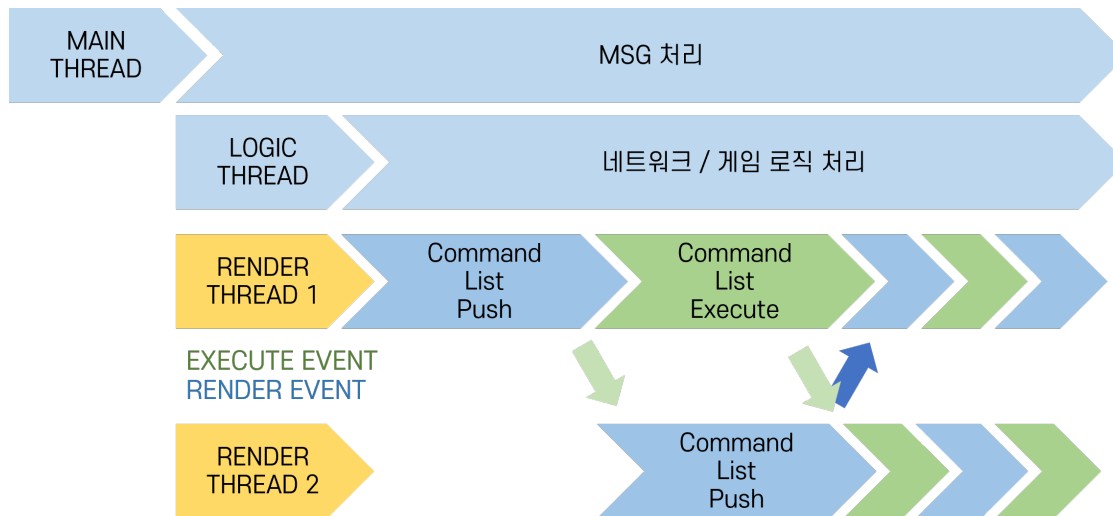
[https://www.youtube.com/watch?v=LqkQf\\_4yFbI](https://www.youtube.com/watch?v=LqkQf_4yFbI)

## DirectX12 멀티쓰레드 렌더링 구현

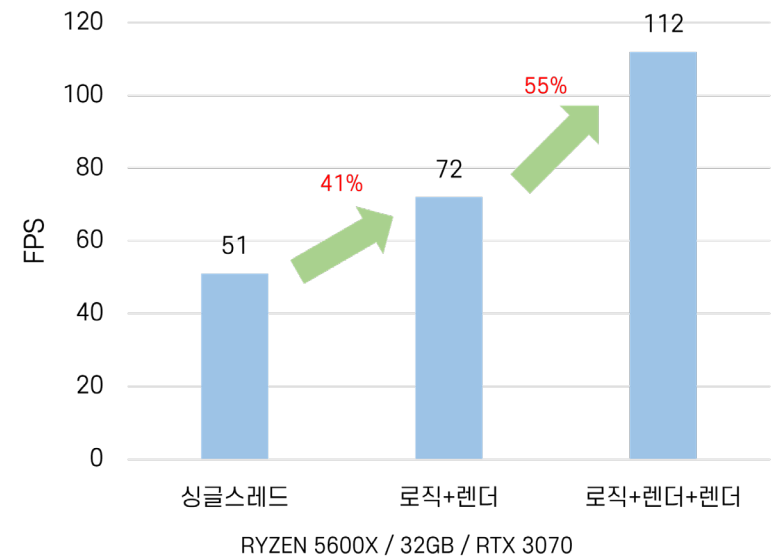
총 4개의 스레드를 사용하여, 클라이언트의 프레임을 향상

Render Thread간의 동기화는 2개의 이벤트를 사용하여 관리

결과, 싱글 스레드 대비 약 110%의 성능 향상



멀티쓰레드 렌더링 동작 소개



프레임 변화 측정



```
thread logical_thread{ &Client::Logic,&client };

thread render_thread1{ &Client::Draw,&client };
thread render_thread2{ &Client::Draw,&client };
```

```
void Draw() {

    cout << "DRAW CALL" << endl;
    int i_now_render_index;
    if (!_render_thread_num) {
        i_now_render_index = 0;
        _render_thread_num++;
    }
    else {
        i_now_render_index = 1;
        _render_thread_num = 0;
    }
    while (g_isLive) {
        ::WaitForSingleObject(dxEngine._renderEvent, INFINITE);

        dxEngine.timerPtr->TimerUpdate();

        dxEngine.timerPtr->ShowFps(windowInfo);

        dxEngine.Update(windowInfo, isActive);

        dxEngine.Draw_multi(windowInfo, i_now_render_index);
    }
}
```

로컬 변수 i\_now\_render\_index 를 사용하여,  
Atomic 변수인 i\_now\_render\_index 에 최소한으로 접근

메소드를 스레드로 분리하여, 내부의 데이터를 사용할 수 있도록 구성

두 개의 스레드는 이벤트로 제어  
각 이벤트는 command list에 push/ Execute 가 된 이후 Set

```
::WaitForSingleObject(_excuteEvent, INFINITE);
SetEvent(_renderEvent);
```

```
D3D12_RESOURCE_BARRIER barrier2 = CD3DX12_RESOURCE_BARRIER::Transition(swapChainPtr->_renderTargets[i_now_render_index].Get(), D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT);
cmdList->ResourceBarrier(1, &barrier2);
cmdList->Close();
```

...

```
swapChainPtr->_swapChain->Present(0, 0);
```

```
cmdQueuePtr->WaitSync();
```

```
swapChainPtr->_backBufferIndex = (swapChainPtr->_backBufferIndex + 1) % SWAP_CHAIN_BUFFER_COUNT;
```

```
SetEvent(_excuteEvent);
```

## 네트워크 최적화를 위한 이동 보간

이동 패킷의 전송량을 낮춤에 따라서, 오브젝트 이동 시 끊김 발생

끊김 방지를 위해서 이동 동기화 개발

직전 패킷부터 현재 패킷까지 프레임별로 이동 양을 제어하여 선형 보간

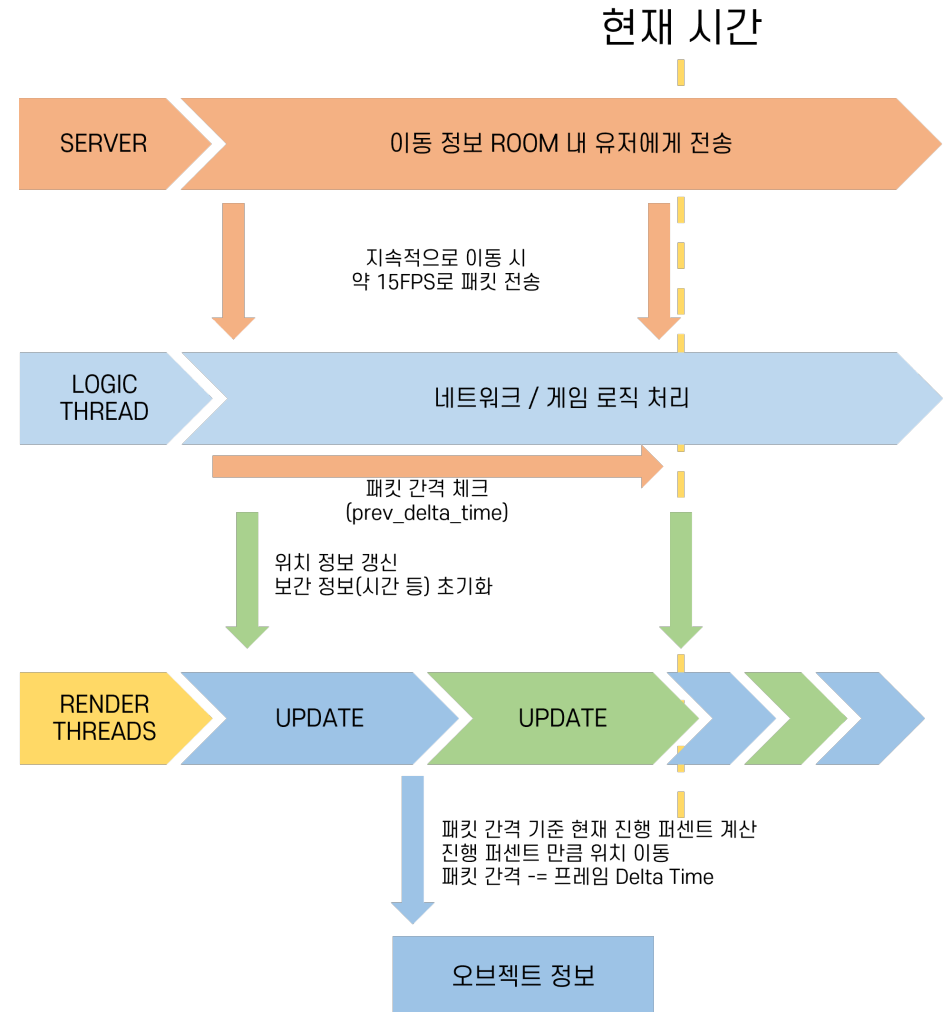
```
for (int i = 1; i < PLAYERMAX; ++i) {
    if (playerArr[i]._on) {
        float dt = timerPtr->deltaTime;
        playerArr[i]._delta_percent = dt / playerArr[i]._prev_delta_time;

        if (playerArr[i]._prev_delta_time > 0) {
            playerArr[i]._prev_delta_time -= dt;
        }
        else {
            playerArr[i]._prev_delta_time = 0;
            playerArr[i]._delta_percent = 0;
        }

        playerArr[i]._delta_transform = playerArr[i]._transform - playerArr[i]._prev_transform;
        playerArr[i]._delta_transform = playerArr[i]._delta_transform * playerArr[i]._delta_percent;

        playerArr[i]._prev_transform = playerArr[i]._prev_transform + playerArr[i]._delta_transform;
        playerArr[i]._prev_degree = playerArr[i]._degree;
    }
}
```

DxEngine.cpp / line :1322~







# 게임서버 텀 프로젝트

[https://github.com/haein0303/GameServer\\_Final\\_term](https://github.com/haein0303/GameServer_Final_term)

## 프로젝트 소개

- IOCP를 활용한 MMORPG 게임 서버 구현

## 구현 내용(C++ / IOCP)

- 시야처리
- 파티
- 경험치 및 레벨
- 채팅
- NPC 로밍
- NPC Agro

## 파티 구현

```
unsigned int party_counter = 0;
concurrency::concurrent_unordered_map<unsigned int, PARTY> party_map;
mutex pm_mu;
```

```
if (clients[pa]._hp <= 0) { //NPC 죽음
    //나중에 재활용할 수 있으니깐
    clients[pa]._target_id = -1;

    clients[pa]._ll.lock();
    clients[pa]._state = ST_ALLOC;
    clients[pa]._ll.unlock();

    clients[c_id].send_die_packet(pa, 100);

    if (clients[c_id]._party_id > -1) {
        auto get_pair = party_map.find(clients[c_id]._party_id);
        for (int i = 0; i < MAX_PARTY; ++i) {
            if (get_pair->second._player_id[i] == -1) continue;
            printf("PARTY BONUS : %d [%d]Wn", get_pair->second._player_id[i], 20);
            clients[get_pair->second._player_id[i]].send_die_packet(c_id, 20);
        }
    }

    for (int pl : l_list) {
        if (is_pc(pl)) {
            clients[pl].send_remove_player_packet(pa);
        }
    }

    TIMER_EVENT ev{ pa, chrono::system_clock::now() + 30s, EV_RESET_NPC, 0 };
    timer_queue.push(ev);
}
```

## 가입과 삭제 보조 함수

배열을 재사용하기 위해서, 추가와 삭제를 담당하는 함수를 작성하였습니다.

## 파티 보너스

만약 오브젝트가 파티 아이디가 있다면, 파티원에게 경험치 일부를 공유합니다.


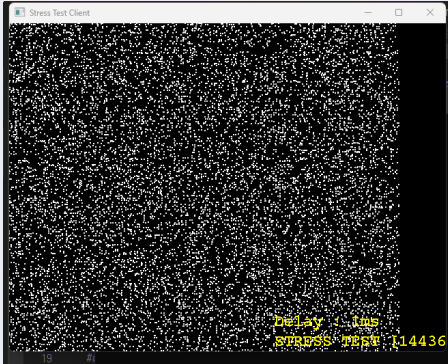
## 관리

파티원은 배열로 관리합니다.

컨테이너를 사용하기에 크기가 정해져 있고, 반복 횟수가 많지 않아서 배열로 처리하였습니다.

```
//조인 가능하면 1, 아니면 0
int join(int id) {
    p_l.lock();
    for (int i = 0; i < MAX_PARTY; ++i) {
        if (_player_id[i] == -1) {
            _player_id[i] = id;
            p_l.unlock();
            return 1;
        }
    }
    p_l.unlock();
    return 0;
}

//성공하면 1, 아니면 0
int exit(int id) {
    p_l.lock();
    for (int i = 0; i < MAX_PARTY; ++i) {
        if (_player_id[i] == id) {
            _player_id[i] = -1;
            p_l.unlock();
            return 1;
        }
    }
    p_l.unlock();
    return 0;
}
```

Multithread IOCP	최적화 이후
	
최대 동시 접속	
653	14436

## 최적화 수행 결과

[https://github.com/haein0303/Game\\_server\\_totutorial](https://github.com/haein0303/Game_server_totutorial)

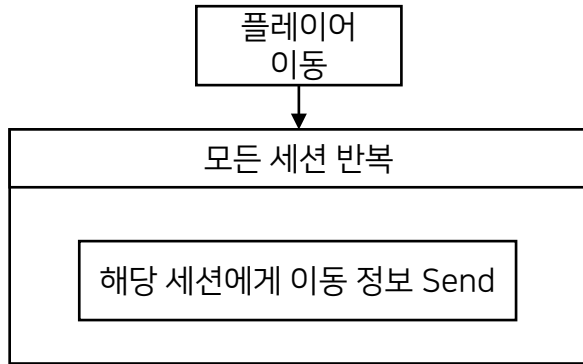
### 구현 의도

모든 세션에게 데이터를 전송하는 Network overhead를 줄이기 위하여, 시야처리 및 ZONE을 구현

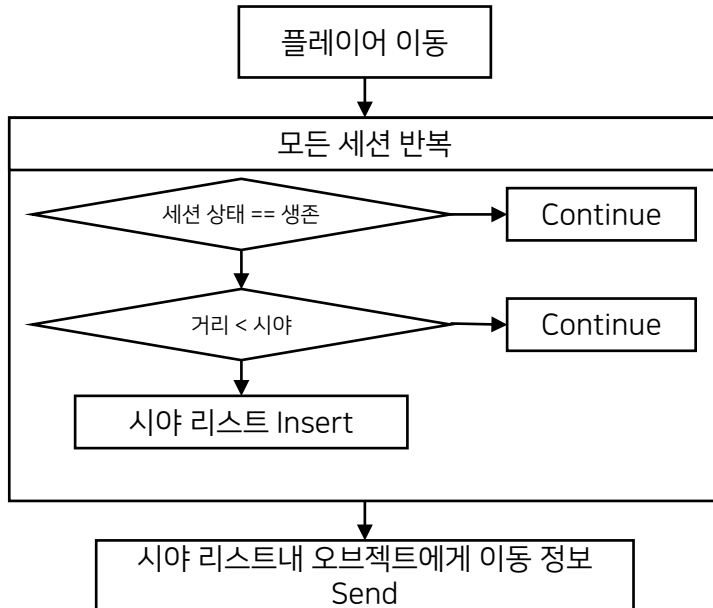
### 구현 내용

시야처리  
ZONE 구현  
Multi Thread IOCP

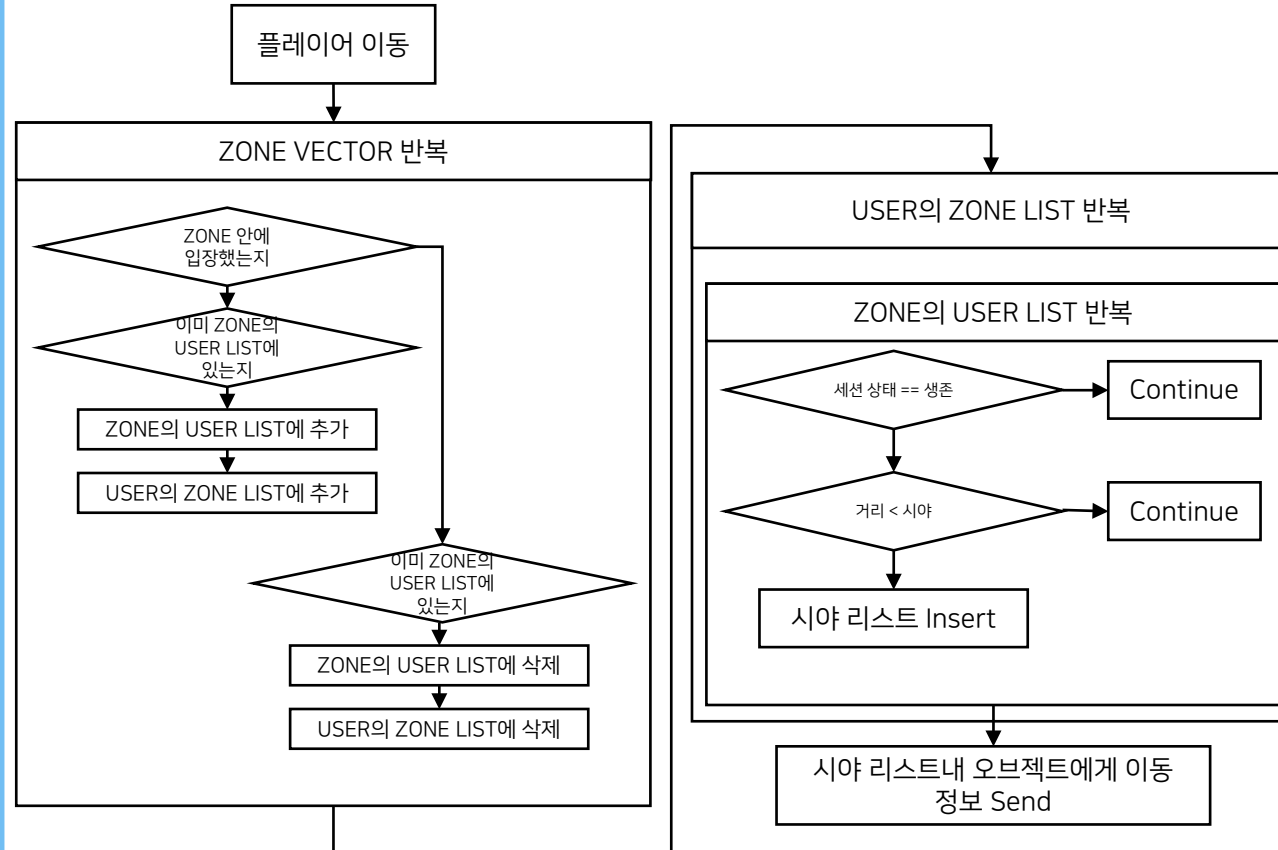
## 기존 MultiThread IOCP 방식



## 시야처리 적용



## 시야처리 및 ZONE 적용



## ZONE 내부에 있는지 검사 및 처리 코드

```
bool contains(SESSION& player) {  
  
    bool is_in = false;  
    //내부에 있는지 검사  
    if (player.x >= x_min_ && player.x <= x_max_ && player.y >= y_min_ && player.y <= y_max_)  
    {  
        is_in = true;  
    };  
  
    if (is_in) {  
        //없을때 처리  
        //없을때만 검사하고 동작시켜야함  
        zl.lock();  
        if (user_list.count(player._id) == 0) {  
            //유저 zonelist에 추가  
            //내꺼도 추가  
  
            user_list.insert(player._id);  
            zl.unlock();  
            player._zl.lock();  
            player._zone_list.insert(my_num);  
            player._zl.unlock();  
            return is_in;  
        }  
        zl.unlock();  
    }  
    else {  
        //있을때 처리  
        //있을때만 검사하고 동작시켜야함  
        zl.lock();  
        if (user_list.count(player._id) != 0) {  
            //유저 zonelist에서 빼줘야됨  
            //내꺼에서도 빼야됨  
  
            user_list.erase(player._id);  
            zl.unlock();  
            player._zl.lock();  
            player._zone_list.erase(my_num);  
            player._zl.unlock();  
            return is_in;  
        }  
        zl.unlock();  
    }  
  
    return is_in;  
}
```

## 검사 이후 시야리스트 갱신 및 전송 코드

```
clients[c_id]._vl.lock();  
auto old_vl = clients[c_id]._view_list;  
clients[c_id]._vl.unlock();  
  
//Todo : 모든 클라이언트 검사가 아니라 zone에서 검사하자  
  
for (auto& m : g_map.zone_list) {  
    m.contains(clients[c_id]);  
}  
  
unordered_set<int> new_vl;  
  
for (auto& vl : clients[c_id]._zone_list) {  
    for (auto& p : g_map.zone_list[vl].user_list) {  
        if (p == c_id) continue;  
        if (can_see(c_id, p) == false) continue;  
        new_vl.insert(p);  
    }  
}  
  
for (auto& o : new_vl) {  
    if (old_vl.count(o) == 0) {  
        clients[o].send_add_player_packet(c_id);  
        clients[c_id].send_add_player_packet(o);  
    }  
    else {  
        clients[o].send_move_packet(c_id);  
        clients[c_id].send_move_packet(o);  
    }  
}  
clients[c_id].send_move_packet(c_id);
```

네트워크 overhead를 최소화 하기 위하여  
시야처리를 도입하였고, 시야처리를 처리하기 위한 연산이  
Overhead가 되기 때문에, ZONE을 활용하여 성능 개선

## 흔한 살아남아 보자

장르 : 3D 디펜스

개발기간 : 2022. 05.

사용도구 : Unity

개발인원 : 1명

Git : [https://github.com/haein0303/survive\\_normal\\_attack](https://github.com/haein0303/survive_normal_attack)

### 게임 소개

무한으로 나오는 다양한 종류의 몬스터를 피하고,  
공격하며 최대한 살아남아 기록을 갱신하는 게임입니다.

### 구현 내용

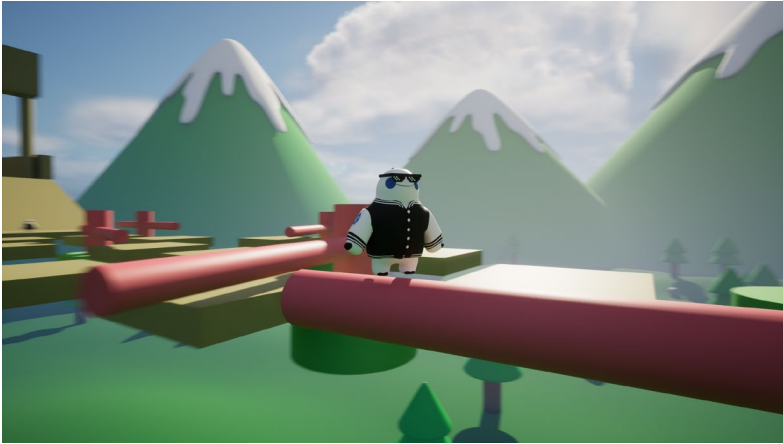
네비게이션 메쉬 활용 NPC 구현

트레일 렌더러 활용 공격체 구현

NPC 원거리 공격 구현







# TIIINO

## 프로젝트 소개

- 언리얼 5를 이용한 온라인 레이싱 파티 게임 제작

장르 : 레이싱, 파티 게임

개발기간 : 2022. 9.19. ~

사용도구 : Unreal5 / IOCP / Github /

개발인원 : 13명

- PD (1) / PM (1) / TD (1)
- 클라이언트 프로그래머 (2) / 서버 프로그래머 (3)
- 기획 (2) / 그래픽 (2) / 사운드 (1)

Git : <https://github.com/sadending82/Tiiino>

역할 : PD

FGT 진행 ( 2023. 10. 13. PM 8:00 ~ 10:20 )

OBT 진행 ( 2023. 11. 14. ~ 16. )

