데이터 동기화 문제 해결에 자신 있는,
서버 프로그래머 김재우 입니다.

C/C++ C# Server

PORTFOLIO

contact@thejaeu.com



김재우

jaeu

Projects

2026.04 - 진행중	**FPS 서버 프로젝트** “오버워치” 네트워크 문제를 겪고 내부 로직을 역설계 한 서버 연구
2025.03 - 2025.07	**Lockstep 동기화 서버 프로젝트** C++ Boost.Asio를 활용한 Native 서버 연구
2024.08 - 2024.11	**졸업 작품** “Dessert Planet” 개발
2023.08 - 2023.12	**학기 프로젝트** “Dream of DeAD” 개발

Education

2025.03 - 2026.02	**청강문화산업대학교** 차세대게임콘텐츠학과 졸업예정 (전공심화과정)
2020.03 - 2025.02	**청강문화산업대학교** 게임전공 졸업
2017.03 - 2020.02	**한세사이버보안고등학교** 해킹보안과 졸업

Skills

Language : C/C++, C#

Server & Network : Socket, Boost.Asio, IOCP

Database : MySQL, Redis

Client/Engine : Unity

Tools : Git, Notion, Markdown

Certificate

2025.08	JLPT N2 - 118점
2018.12	정보처리기능사

프로젝트 소개

서버 프로젝트

Lockstep 동기화 서버 연구 프로젝트

Boost.Asio를 통해 Lockstep 동기화를 진행하는 서버를 직접 설계했습니다
https://thejaeu.com/server-research
https://youtu.be/igMT_L6Az30

FPS 서버 프로젝트 (진행중)

FPS 장르의 서버 아키텍처를 예상 및 역설계 하여 적용 해 보았습니다
https://thejaeu.com/fps-reverse-planning-server-research

팀 프로젝트

학기 프로젝트 “Dream of DeAD”

프로그래밍 파트장을 맡았으며, 협업 프로세스를 경험 할 수 있었습니다
프로젝트 : https://thejaeu.com/xr-contents-project
프로젝트 영상 : https://youtu.be/hsBnVk2Jzal

Lockstep 동기화 서버 연구 프로젝트

Boost.asio Lockstep AI 활용

https://thejaeu.com/server-research
https://youtu.be/igMT_L6Az30

Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

핵심 키워드

▼ 프로젝트 설계

- 그룹 간 위치 동기화를 위해 Lockstep 알고리즘을 선택하였고,
  스텝 별 정밀한 위치 동기화를 진행할 수 있도록 최적화 하였습니다
- Lockstep을 수행하기 위한 각 역할을 고려하여 서버를 설계하였습니다

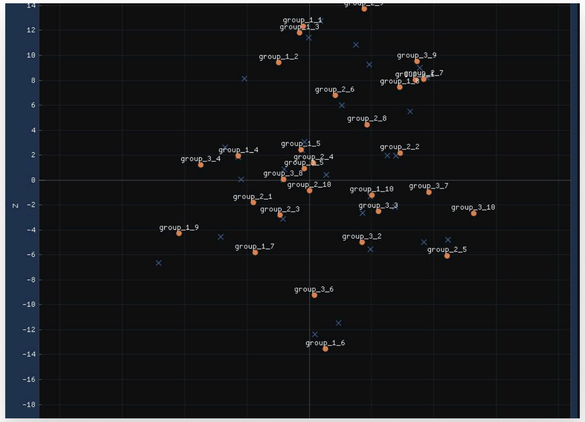
▼ 이슈 트래킹과 해결

- 프로젝트 진행 과정에서 서버 로직 구현 등에서 문제가 여러 발생하였고,
  그것들을 해결해 나아가며 새로 알게 되거나 복습하게 된 지식으로
  프로젝트를 보완할 수 있었습니다

▼ AI 테스트 클라이언트

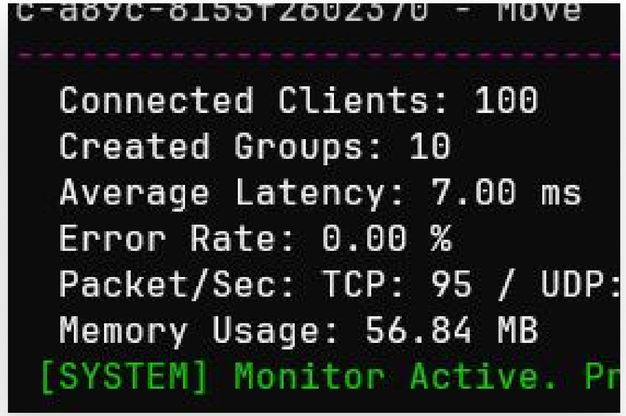
- 테스트 클라이언트의 작성에 많은 시간을 할애할 수 없었던 상황과
  반복적인 작업을 빠르게 수행하기 위해 AI 를 활용하여
  기존에 눈에 잘 보이지 않던 서버 데이터들을 시각화 하였습니다

주요 성과



- **여러 클라이언트의 스텝별 위치 동기화**

  AI를 통해 제작한 테스트 클라이언트로
  여러 클라이언트를 TCP 연결을 통해
  서버와 정해진 프로토콜로 통신하여
  각 그룹에 맞는 스텝 별 위치 동기화를 달성



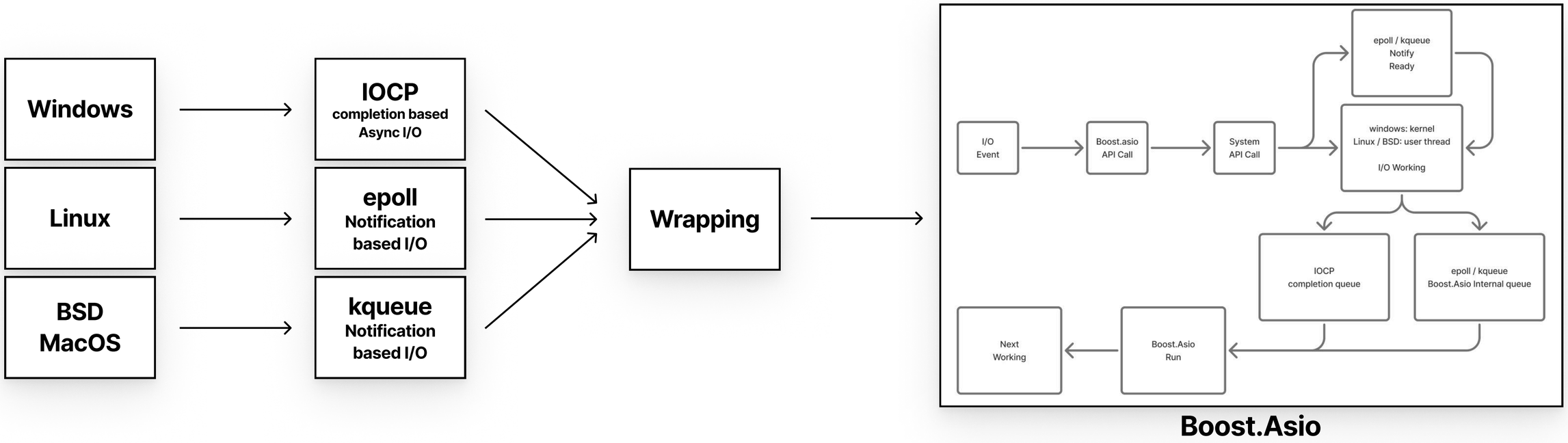
- **서버의 처리 능력**

  각 클라이언트를 그룹으로 나누고
  해당 그룹내에서 위치 동기화를 진행
  100개의 클라이언트를 10개씩 10개의 그룹으로
  나누어 처리한 결과
  평균 지연 속도 약 7ms
  오류 0% (Udp 처리 손실)
  메모리 사용량 100MB 미만 달성

Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

프로젝트 설계 1

- **네트워크 라이브러리로 Boost.Asio를 선택한 이유**
  어느 환경에 서버가 구동될 지 모르는 상황에서 OS에 종속적이지 않아야 한다고 판단하였습니다
  비동기 I/O를 활용할 수 있어야 하며 각각 IOCP, epoll, kqueue의 사용을 우선시 하였습니다
  각 OS 별로 비동기 I/O 모델을 고수준으로 래핑한 Boost.Asio를 알게 되었습니다
  신뢰도 높은 Boost 라이브러리이며 Asio 자체 스케줄러를 통한 고성능 처리가 가능하다 판단하여 최종 결정할 수 있었습니다



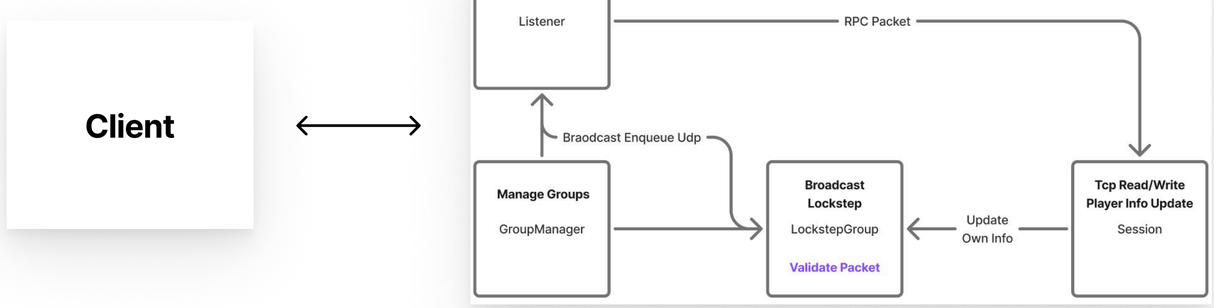
Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

프로젝트 설계 2

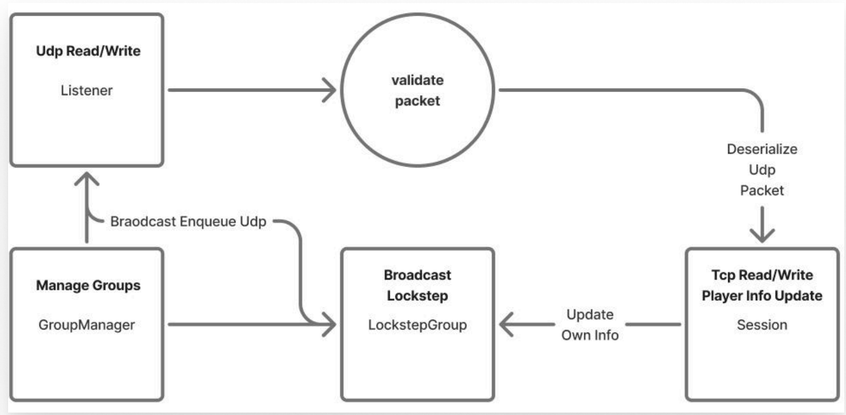
- **패킷 프로토콜과 서버 권위적 처리**
  클라이언트의 입력에 대해 빠르고 간결한 전달이 필요하여 Json, XML, Binary 등 직렬화 방법을 고민했습니다
  결론적으로 사용성이 높으며 각 언어에 대한 대응이 가능한 Binary 기반 Protobuf 직렬화를 최종적으로 선택할 수 있었습니다

  서버 내부처리는 Accept와 Udp 통신을 담당하는 Listener, 각 클라이언트의 Tcp 통신을 담당하고 플레이어 정보를 가지는 Session
  Lockstep과 그룹 별 상태를 저장하고 업데이트하는 LockstepGroup, 해당 LockstepGroup을 관리하는 LockstepGroupManager로 구성하였습니다

  LockstepGroup은 Bucket(한 스텝 단위)별로 Input을 저장 및 Broadcast 진행하고,
  Attack 패킷에 대해서는 Validate를 진행하여 부정한 패킷을 방지하였습니다



Protobuf 기반 직렬화



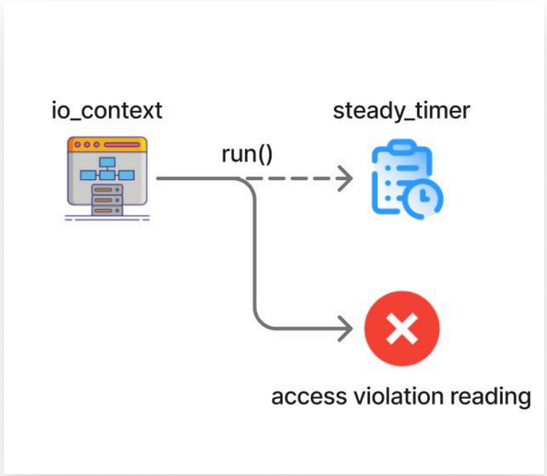
서버 내부 처리 과정

Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

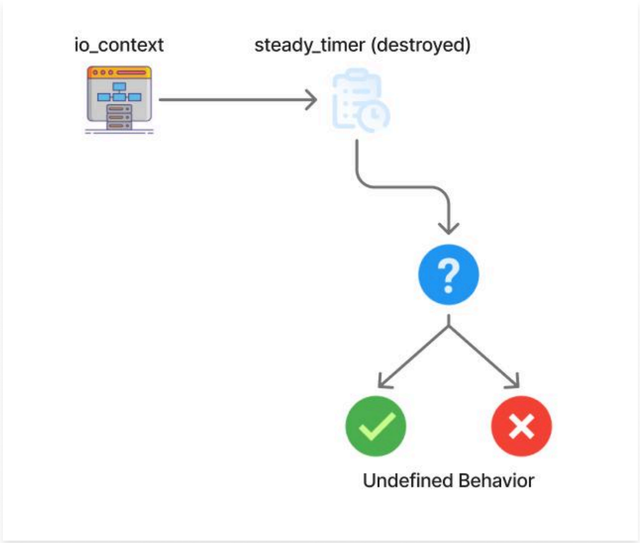
이슈 트래킹 1

- **잘못된 캡처로 인한 치명적 버그**
  steady_timer를 활용하여 Scheduler 클래스를 구현 및 사용하는 과정에서 발생한 dangling pointer 오류로 다음과 같은 문제해결 과정을 겪었습니다

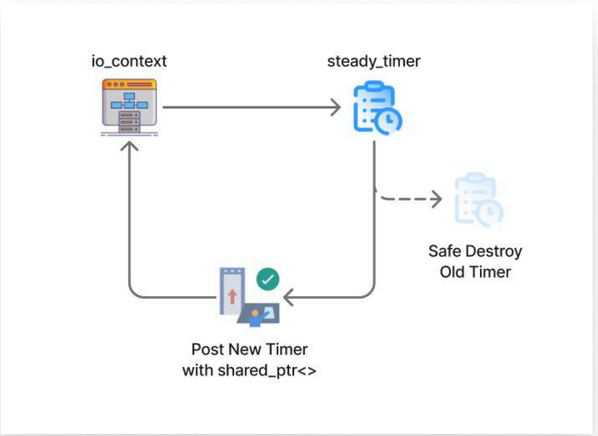
- **개요**
  - 목적 : steady_timer를 통해 주기적인 작업 실행
  - 구현 : 해당 타이머 기반의 Scheduler 클래스 설계 및 적용
  - 문제 : 간헐적인 Access Violation Reading 오류 발생



- **원인 분석**
  1. [this] 캡처 실수
     - async_await() 콜백 등록 시 [this] 사용
     - this는 Raw Pointer로 참조 카운트 증가 없음
  2. 수명 주기로 인한 dangling pointer 참조
     - 타이머 대기 중 외부에서 Scheduler 소멸
     - 그로인한 dangling pointer 참조 및 UB 발생



- **해결 방법**
  - 람다에서 shared_from_this()로 얻은 shared_ptr 캡처
  - 비동기 작업 완료 시 까지 참조 카운트를 유지하여 Scheduler의 수명 보장

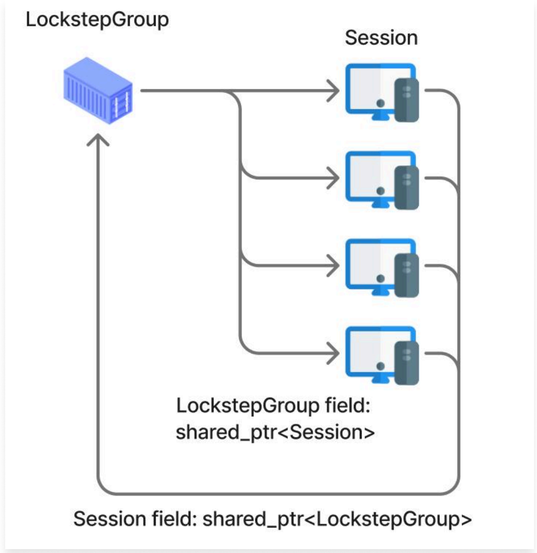


Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

이슈 트래킹 2

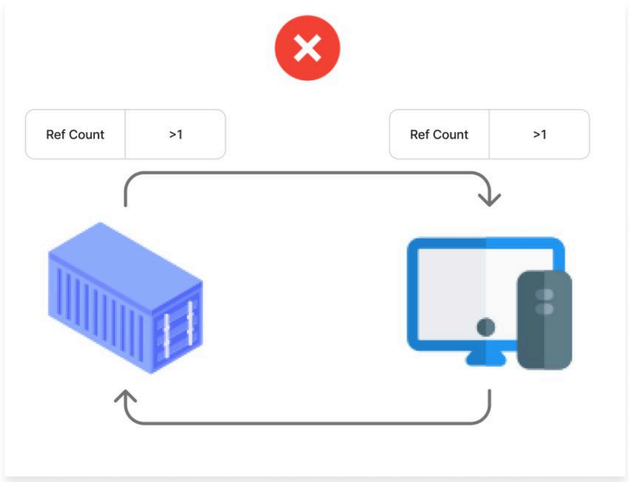
- **순환참조로 인한 메모리 해제 불가**
  shared_ptr로 객체의 수명을 관리하는 과정에서 발생 한 이슈로
  다음과 같은 문제 해결 과정을 겪었습니다

- **개요**
  - 목적 : 스마트 포인터를 통한 RAII 패턴의 메모리 관리
  - 구현 : 각 클래스 내의 멤버 필드로 shared_ptr을 통한 객체 수명 관리
  - 문제 : 해제되어야 할 메모리가 해제되지 않아 메모리 누수 및 프로그램 종료 불가

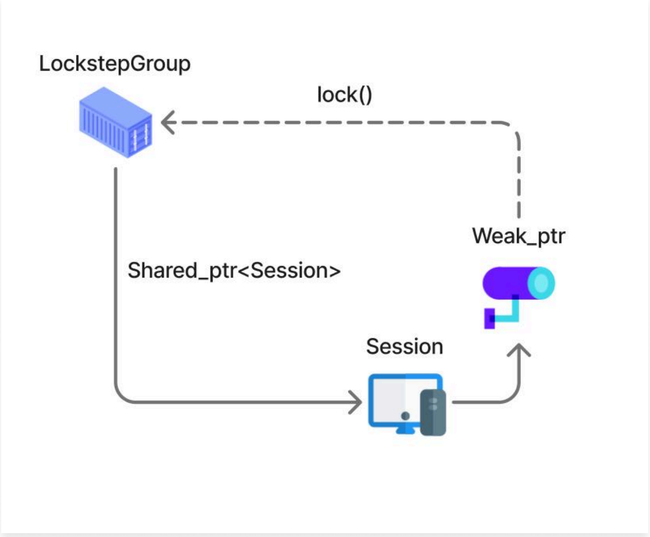


- **원인 분석**
  1. 컨테이너로 여러 Session 객체를 관리하는 구조에서 Session 객체가 상위 객체를 참조하는 상황 발생 (상위 객체 특정 함수 실행 등)
  2. 같은 구조에서 여러 상위 객체가 하위 객체와 상호 참조하는 상황이 반복 발생

  - 참조 카운트가 0이 되지 않아 자동해제가 발생하지 않음



- **해결 방법**
  - 소유권을 명확히 하여 관리 객체만이 하위 객체를 소유하고 다른 객체가 접근 시 weak_ptr 참조를 통해 소유권 분리 및 안전한 접근 설계
  - weak_ptr 참조 중인 객체에 접근 시 lock()으로 유효성 확인 후 접근



Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

AI 를 활용한 빠른 테스트 작업

학교 수업 연구 프로젝트 총 기간 (약 9개월)							
1학기				2학기			
연구 주제 선정	시스템 설계	평가 준비	이슈 트래킹 (버그 수정 등)	실제 구현	Unity Client 제작	진행 보고서 작성	외부 행사 참여 (unite 2026 등)
기술 스택 조사	실제 구현	진행 보고서 작성	시스템 재설계	이슈 트래킹 (버그 수정 등)	평가 준비	데이터 시각화 고민	AI 활용 Client 제작 (*기간 단축)

- **AI 활용에 따른 시간 단축과 데이터의 시각화**

  위의 표는 전반적인 프로젝트의 기간 별 진행 작업 등을 나타낸 것입니다
  실제로 이슈 트래킹과 클라이언트 제작 등에 많은 시간이 소모 되었으며 시간 투자에 비해 결과는 만족스럽지 않았습니다

  이 때, 활용하고자 한 것은 AI 였습니다

  기존에도 사용을 안하고 있던 것은 아니지만 CLI 를 통해 코드에 직접 관여하는 방식은 프로젝트 후반에 다 와서 실행 해보게 되었습니다

  Gemini CLI 를 사용하여 서버 API에 대한 클라이언트를 제작하도록 명령하였고, 데이터의 시각화에 관한 고민도 해결할 수 있었습니다
  AI 덕분에 클라이언트 제작 기간을 획기적으로 단축할 수 있었고, 서버 시스템 중 일부 문제들도 해결할 수 있었습니다

↓

Client with AI Github

Lockstep 동기화 서버 연구 프로젝트에서 이런 경험을 쌓았어요

Lockstep 동기화 서버 연구를 통해 얻은 것

적절한 아키텍처의 설계

- 필요한 기능과 상황을 빠르게 파악하고 적용할 기술 및 라이브러리를 직접 설계해볼 수 있었습니다
- 직접 설계를 통해 효율적인 구조를 찾아나가면서 서버 시스템을 구현해볼 수 있었습니다

이슈 트래킹의 중요성

- 이슈 트래킹을 통해 고질적인 문제 혹은 미처 생각하지 못했던 문제를 직접 해결해볼 수 있었습니다
- 단순 이슈의 해결에 그치지 않고 재발 방지를 위한 회고 등을 통해 지식을 넓힐 수 있었습니다
  - 스마트 포인터 순환 참조 블로그 글 - https://blog.thejaeu.com/c++/smart-pointer/

AI를 통한 아키텍처 확인 및 테스트 환경 개발

- AI의 성능이 증가함에 따라 문제점을 간접적으로 파악하면서 아키텍처 설계의 문제 발생 가능성에 대해 넓은 시야를 가질 수 있었습니다
- AI의 빠른 분석과 코드 작성으로 인해 클라이언트 제작의 시간을 단축하여 서버의 테스트 환경을 빠르게 조성할 수 있었습니다

FPS 서버 프로젝트(프로젝트 진행중)

FPS Rewind Dedicated

https://github.com/thejae-u/fps-reverse-planning-server-research 

FPS 서버 프로젝트에서 이런 경험을 쌓았어요

핵심 키워드

▼ 지연 보상 시스템

- 네트워크 지연에 의한 동기화 문제가 발생했고,
  해결하기 위한 Rewind 시스템을 도입했습니다
- Rewind 시스템은 스냅샷을 기반으로 입력 당시 서버 상태에 의해
  동기화 처리를 합니다

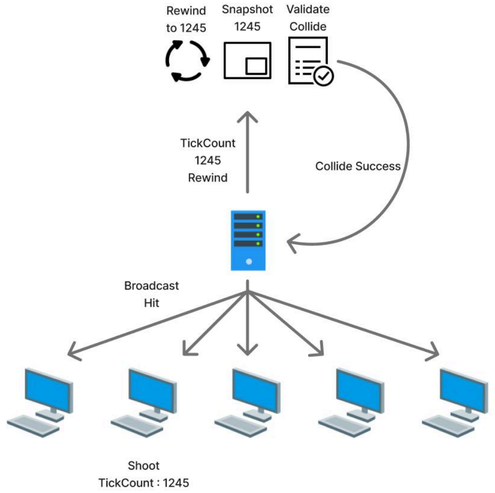
▼ 완전한 서버 권위적 처리

- 서버 내의 World 를 통해 클라이언트의 상태를 유지하며,
  클라이언트의 입력에 의해 World를 업데이트 합니다
- 클라이언트는 입력만을 전송하며
  서버 측에서 검증 및 연산을 수행하여 클라이언트에게 통보합니다

▼ Dedicated Logic Server

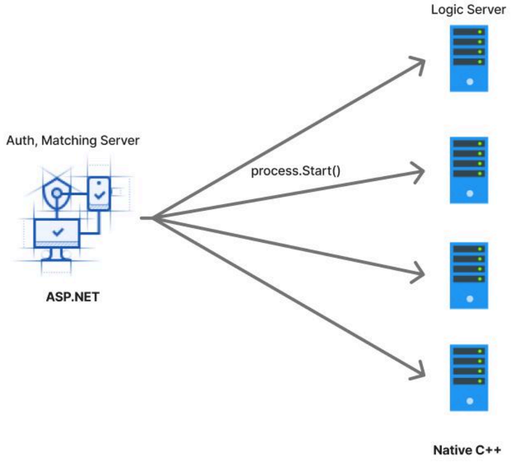
- 기존 Lockstep 서버의 문제점인 싱글 프로세스, 멀티 스레딩 구조를
  해결하기 위해 도입한 방식입니다
- ASP.NET 인증 서버를 통해 Dedicated 서버를 실행하며,
  다중 프로세스 구조를 가지게 됩니다

주요 성과



Rewind System

- 입력 당시 Snapshot으로 되돌리는 Rewind
- Snapshot을 기반으로 충돌 체크



Dedicated Server

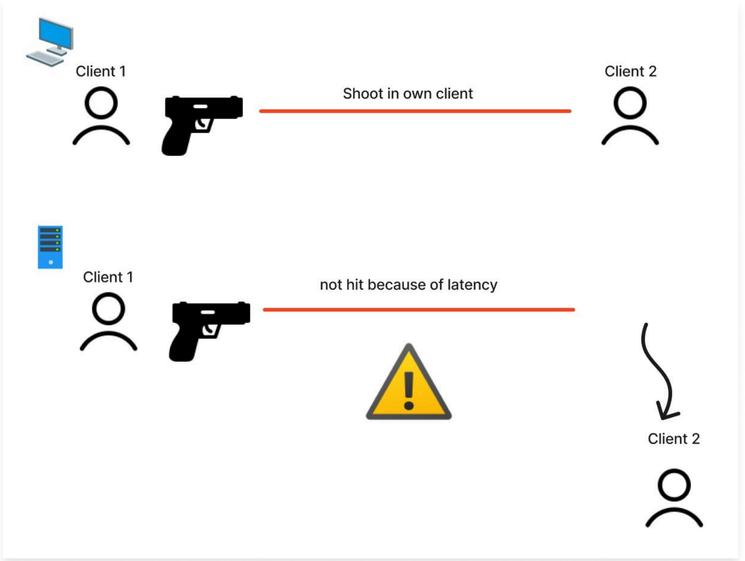
- 인증과 매칭을 담당하는 ASP.NET 웹 서버
- 매칭이 완료되면 각종 서버 정보와 함께
  Process.Start()
- 게임 종료 이후 웹 서버에 필요한 정보를
  전송하고 소멸하는 로직 서버

FPS 서버 프로젝트에서 이런 경험을 쌓았어요

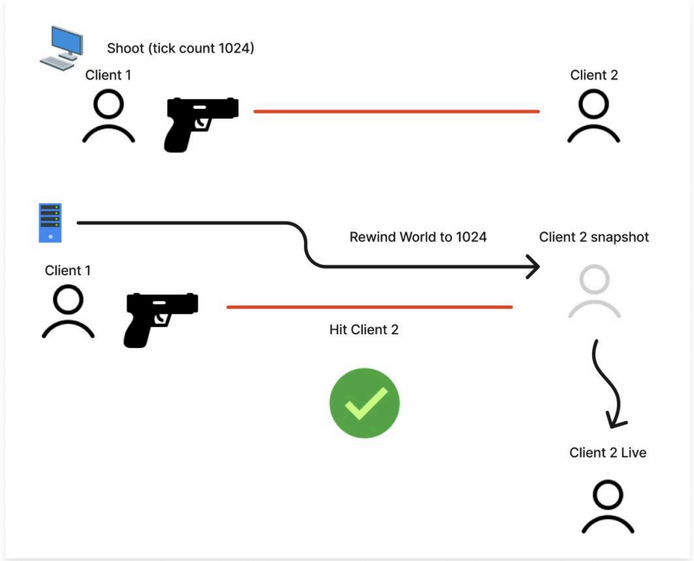
지연 보상 시스템 Rewind

네트워크 지연 문제로 인한 클라이언트 입력이 정확한 판정을 기대할 수 없는 문제가 있었습니다
Rewind & Restore 시스템을 도입하여 Snapshot을 기반으로 과거 위치 정보를 일정 수준 기록 하여
지연 시간 만큼의 Rewind 상태에서 충돌 판정 후, 충돌에 대한 Hit 패킷 생성까지 원자적으로 처리했습니다

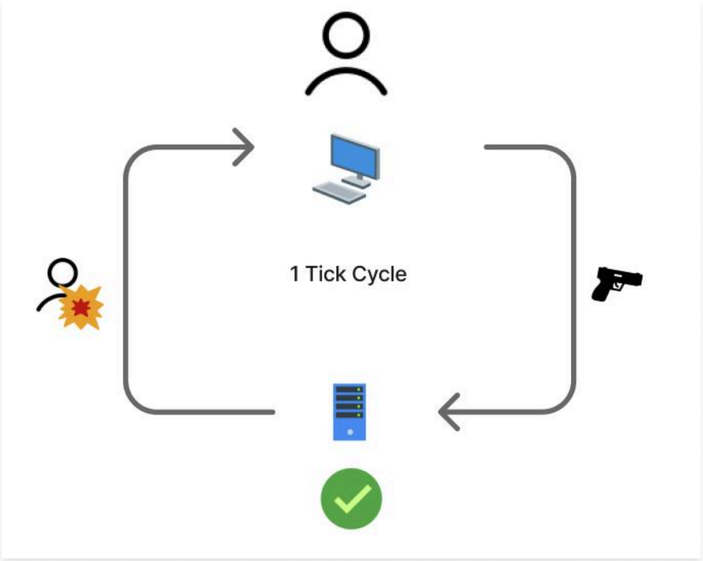
- **기존 시스템**
  - 플레이어 간 Shoot & Hit 판정이 불가능한 문제 발생
  - 네트워크 지연 해소 로직 부족



- **Snapshot 기반의 Rewind 시스템**
  - 클라이언트의 Shoot 입력에 tick count를 포함
  - 서버는 해당 가장 가까운 snapshot으로 world를 되돌림
  - 기록된 snapshot을 통해 client 2 의 Hit 판정



- **가장 최신 데이터에 영향을 주지 않는 처리**
  - Rewind의 처리는 별도의 컨테이너를 통해 진행
  - Rewind부터 Hit 처리까지는 단일 틱에 원자적으로 완료됨

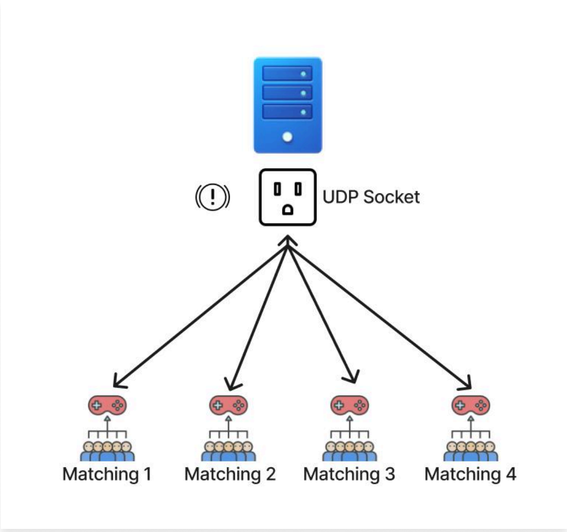


FPS 서버 프로젝트에서 이런 경험을 쌓았어요

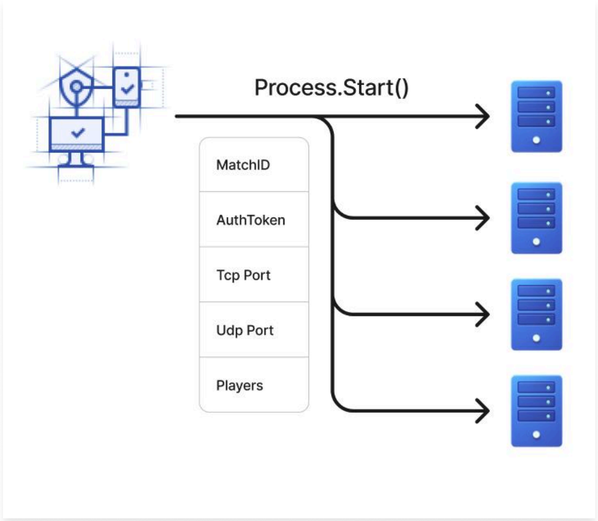
Dedicated Logic Server

Lockstep 동기화 서버의 가장 큰 문제점이었던 싱글 프로세스, 멀티 스레딩 구조의 UDP 통신 과부화 등을 개선하기 위하여
멀티 프로세스, 멀티 스레딩 구조의 Dedicated 서버로 재설계했습니다
그에 따라 인증 서버를 필요로 하여 ASP.NET 을 활용하여 인증 및 매칭 로직을 담당하도록 설계했습니다

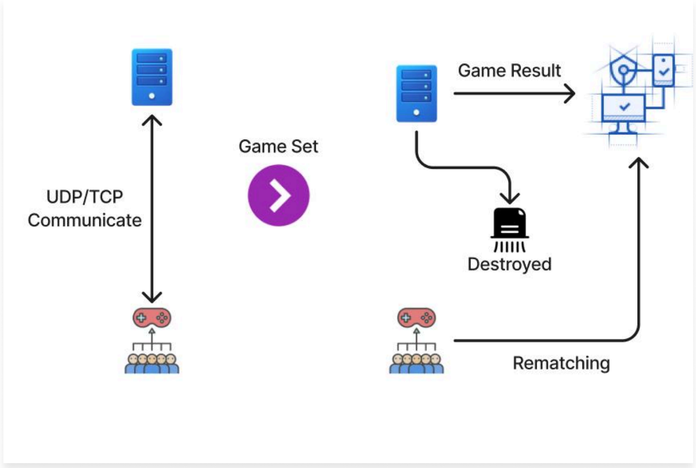
- **싱글 프로세스 시스템의 문제**
  - UDP 소켓을 하나만 사용하여
    네트워크 I/O가 증가하면 밀림 혹은 지연 발생
  - 멀티 스레딩으로 해결하고자 했으나
    Context Switching을 유발하여 오히려 처리지연 증가



- **ASP.NET과 멀티 프로세싱**
  - C# 웹 서버 프레임워크 ASP.NET을 활용하여
    인증 및 유저 매칭 로직을 담당
  - 실행 인자로 필요한 정보 전달
  - 매칭 별 독립적인 프로세스 실행



- **로직 서버의 독립**
  - Socket 기반의 FPS 게임 로직 수행
  - 종료 시 인증서버로 필요한 정보 전송
  - 작업이 완료되면 프로세스 소멸
    OS에 의해 메모리 자동 정리



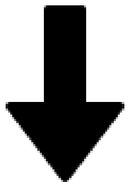
FPS 서버 프로젝트에서 이런 경험을 쌓았어요

FPS 서버에 관한 회고

- **오버워치에서 시작한 프로젝트**
  오버워치 플레이 도중 네트워크 지연 문제에 의한 클라이언트 현상을 보고 궁금증에서 시작하여
  서버 내부의 로직이 어떻게 처리되는지 역으로 설계하여 FPS 게임 서버를 만들고자 했습니다



- **이전 프로젝트의 문제점을 한 단계 개선한 프로젝트**
  Lockstep 동기화 서버 구조인 싱글 프로세스, 멀티 스레딩의 단점인 Udp 과부화,
  Context Switching 등 문제를 파악하고 해결하기 위한 방법을 모색했으며
  실제로 멀티 프로세싱이 가능한 구조의 서버를 적용해 보았습니다



- **앞으로 해야할 일이 남은 프로젝트**
  아직 미완성인 프로젝트로 클라이언트 측 간단 플레이 구현, 네트워크 보간 등 과제가 남았으며,
  컨텐츠 추가, 최적화 등 개선의 여지가 있는 프로젝트 입니다



- **느낀 점**
  단순 호기심으로부터 시작 된 프로젝트지만 FPS 서버의 중요한 지연 문제의 해결과
  이전 프로젝트의 문제점을 개선한 버전의 서버를 설계하고 구현할 수 있던 점에서 성장을 느낄 수 있었습니다

학기 프로젝트 “Dream of DeAD”

Unity 2D Platform Team Project

프로젝트 : https://thejaeu.com/xr-contents-project
프로젝트 영상 : https://youtu.be/hsBnVk2Jzal

PC게임 Dream of DeAD 에서 이런 경험을 쌓았어요

게임 소개

Dream of DeAD는 15주 동안 진행한 Unity 기반 2D 플랫포머 슈팅 프로젝트입니다
플레이어가 스테이지 내 몬스터를 처치하며 목표 지점에 도달하는 PC 게임으로, 13명으로 구성된 기획·프로그래밍·그래픽 파트가 협업하여 개발했습니다
저는 프로그래밍 파트장으로 참여하여 적AI, 사운드, 이펙트, 스테이지 진행 등 주요 시스템을 구현했습니다
또한 프로그래밍 업무를 분담하고, Git 협업 규칙 수립과 타 파트와의 기술 협의를 담당했습니다



플레이 화면

PC게임 Dream of DeAD 에서 이런 경험을 쌓았어요

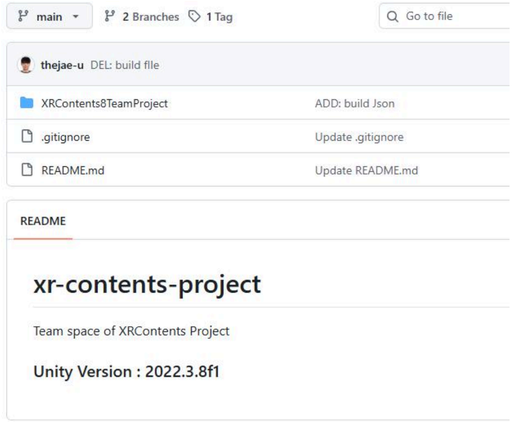
핵심 키워드

▼ 프로그래밍 리딩
- 게임 시스템 설계 주도
- 프로그래밍 업무 분배
- 팀 내 가이드 문서 작성

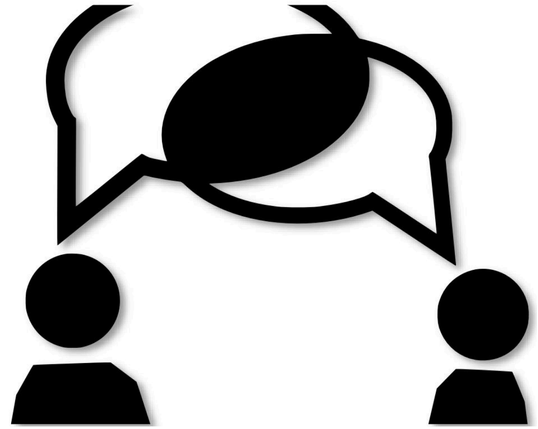
▼ 파트 간 기술 협의
- 그래픽 파트와의 엔진 내 작업 협의
- 기획 파트와의 시스템 구현 방향 협의

▼ 협업 프로세스 관리
- Git 및 네이밍 규칙 정립
- 프로젝트 파일 관리

주요 성과



협업 프로세스 관리
- 프로젝트의 전체 파일 관리 담당
- 팀원들과 네이밍 규칙 공유
- Git 사용 방법 온라인 강의



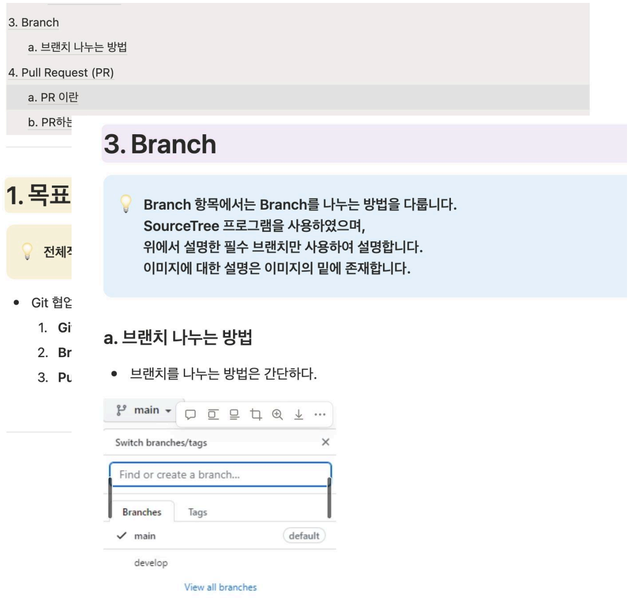
프로그래밍 파트장으로서 역할 수행
- 기획자와의 긴밀한 협업
- Spine 애니메이션, 2D 라이팅 적용 협의
- 프로그래밍 파트 담당 시스템 및 업무 분담 설계

PC게임 Dream of DeAD 에서 이런 경험을 쌓았어요

협업 프로세스 관리

- 13명의 팀원 중 Git 사용 경험이 부족한 인원이 많아 브랜치 생성, 커밋, 병합 과정을 설명하는 가이드 문서를 직접 작성하였습니다
  문서만으로 이해하기 어려운 팀원을 위해 Discord를 통해 실제 저장소를 활용한 실습을 진행했습니다
  이후 팀원들이 정해진 커밋 규칙에 따라 작업할 수 있도록 협업 기준을 마련하였습니다

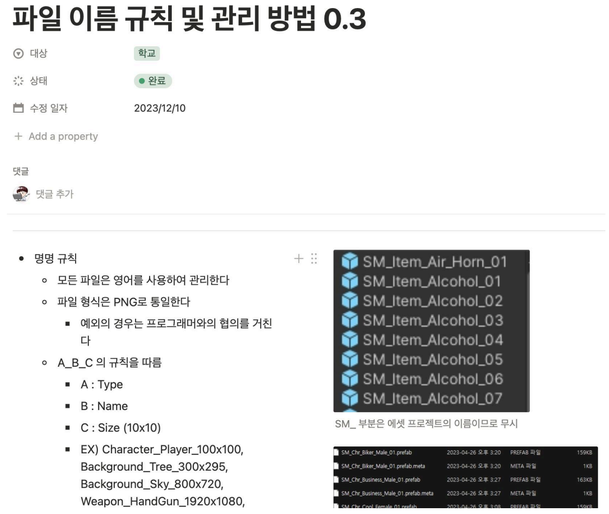
• 팀원을 위한 가이드 문서



Git 사용 가이드 배포 링크 1

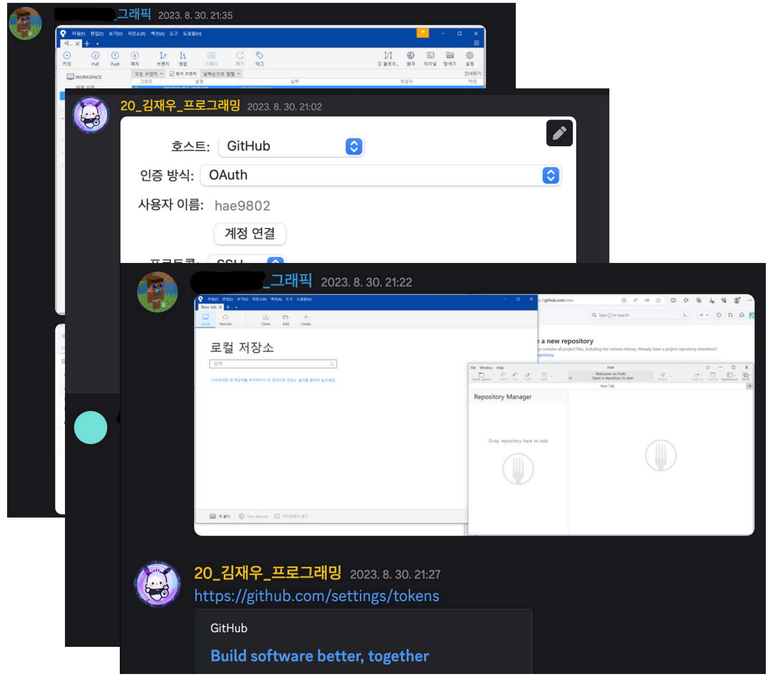
Git 사용 가이드 배포 링크 2

• 네이밍 규칙 규정 및 배포



네이밍 규칙 배포 링크

• 온라인을 통한 버전 관리 툴 사용법 강의



PC게임 Dream of DeAD 에서 이런 경험을 쌓았어요

프로그래밍 파트장으로서의 역할

타 파트 팀원과의 협의

- 기획자 파트와 구현 가능 범위를 공유하고 이를 기준으로 작업 범위를 조정
  - 적 AI의 FSM 구조를 구체화하고, 프로젝트 기간과 인원 내에서 구현 가능한 적 행동 등을 기획자와 협의했습니다
- 그래픽 파트의 Spine 애니메이션 작업, 라이팅 작업 협의
  - 그래픽 파트와 Spine 애니메이션 호출 방식 및 Unity 2D 라이팅 적용 방식을 협의하고, 애니메이션 네이밍 규칙을 정리했습니다.

프로그래밍 업무 분배 방식

- 프로그래머 인원은 13명 중 2명으로 적은 인원이 배치되어 분담이 필요한 상황
  - 시스템 단위로 담당 영역을 나눠 각자의 업무에 집중할 수 있는 전략으로 분담하였습니다

- 서브 프로그래머 : Player Character 시스템
- 프로그래밍 파트장 : 적 AI, 사운드 · 이펙트 시스템 등 기타 핵심 시스템
  - 담당 시스템과 책임 범위를 사전에 구분하여 업무 중복과 담당 영역의 불명확성을 줄였습니다

PC게임 Dream of DeAD 에서 이런 경험을 쌓았어요

프로젝트 진행에 대한 회고

게임 프로젝트 결과

- 게임은 개발 과정 중 주요 검토 대상이었던 조작감·타격감·시각적 피드백을 반영한 최종 빌드를 제작하고 기말 시연을 완료하였습니다
- 최종 시연에서 교수님의 플레이와 피드백을 받았으며, 후속 졸업작품으로 프로젝트를 확장해보자는 제안을 받았습니다

아쉬웠던 점

- 사운드와 이펙트 재생 시마다 인스턴스를 생성하고 제거하는 구조를 사용했으며,
  반복 생성에 대한 최적화를 충분히 적용하지 못했습니다

  이후 다른 프로젝트에서는 Object Pool을 적용하여 객체를 재사용하는 구조로 개선했습니다
  이를 통해 반복적인 생성·삭제 비용을 줄이는 방법을 익혔습니다

- 작업 기준에 대한 의견 차이가 발생했을 때 초기 단계에서 충분히 조율하지 못해,
  이후 작업을 일시적으로 중단한 뒤 별도의 협의를 진행해야 했습니다
  이 경험을 통해 구현 전에 책임 범위와 작업 기준을 합의하는 과정이 중요하다는 점을 배웠습니다

  이후 다른 프로젝트에서는 갈등의 초기 단계를 빠르게 파악하고 책임 범위와 작업 기준을 조율하는 방식으로 개선했습니다

최종 평가

- 처음으로 13명의 인원이 참여하는 대형 프로젝트를 진행하며 프로그래밍 역량 뿐만 아니라
  협업에 대한 역량 또한 크게 성장할 수 있는 계기가 되었습니다

C++/C# 부터 데이터베이스 까지,
안정적인 서버 아키텍처를 설계하고 구현하는 개발자

01

C++/C# 기반 고성능 서버 개발 역량

비동기 I/O의 처리와 각 언어에 대한 이해를 통해
수준 높은 성능의 서버를 개발 할 수 있는 역량을
갖췄고, 스마트 포인터 등의 Modern C++
기술을 보유 했습니다
C# 또한 Unity 엔진 사용, 서버 제작을 통해
중급 이상의 기술을 보유하고 있습니다

02

데이터 기반 문제 해결 및 최적화 능력

게임 데이터에 대한 이해와 서버의 동작 방식을
이해하고 문제가 생겼을 때
창의적이고 효율적인 해결 방법으로
문제를 해결 했던 경험을 통해
서버 개발 중 발생할 수 있는 다양한 문제들에
대응 할 수 있는 역량을 갖췄습니다

03

안정적인 서비스 아키텍처 설계 능력

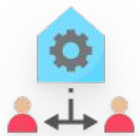
필요한 기능에 대한 처리속도를 높이고자
서버를 직접 설계하고 기술을 선택하며
요구사항을 충족할 수 있는 역량을 갖췄으며
서버의 처음부터 끝을 알고 설계할 수 있는
게임 서버 개발자 입니다

함께 성장하며 팀에 기여하고 싶습니다



빠른 사내 문화 이해 및 적응

팀원으로서의 합류는
사내의 문화를 이해하고 적응
하는 것 부터 시작입니다
입사 시 누구보다 빠른
사내 문화 적응을 보이겠습니다



팀 성장을 위한 발전

합류 후 팀의 성장을 위해
현재에 안주하지 않고,
새로운 기술과 지식을 적극적으로 학습하여
팀에 적극적으로 기여하도록 하겠습니다



팀원과의 커뮤니케이션

앞서 경험했던 소통을 기반으로
팀 내 친목을 도모하고
공적으로나 사적으로나
도움이 될 수 있는 팀원이 되겠습니다

동료와 소통하고 함께 성장하는
서버 프로그래머, 김재우 입니다

- 감사합니다 -

PORTFOLIO

contact@thejaeu.com