클라우드 게이밍 시스템 개발

팀명: 금정산삼고라니

201724579 정현모

201724539 이재욱

201724565 전설

부산대학교 전기컴퓨터공학부 정보컴퓨터공학전공

2022년 8월 01일

지도교수: 김 원 석 (인)

목 차

I. 요구 조건 및 제약사항 분석에 대한 수정사항

i. 요구 조건

본 졸업과제에서 교육 플랫폼 주제로 클라우드 게이밍 시스템을 제작을 목표로 한다. 사용자의 하드웨어 사양에 관계없이 브라우저와 네트워크 환경만으로 동일한 게임 환경을 제공한다. 이를 위하여 Unity Render Streaming 기술과 WebRTC 기술을 활용하여 클라우드 환경 상으로 서비스를 제공하는 것이 목표이다.

ii. 수정 사항

기존에 프론트 부분을 React 프레임워크로 계획하였다. 프론트 이외의 서비스 품질에 집중하기 위해 프론트 부분은 React가 아닌 js와 같은 정적 파일로 작성한다.

II. 설계 상세화 및 변경 내역

클라우드 게이밍 서비스 구축을 위해 크게 4개 파트로 나누어 진행하였다.

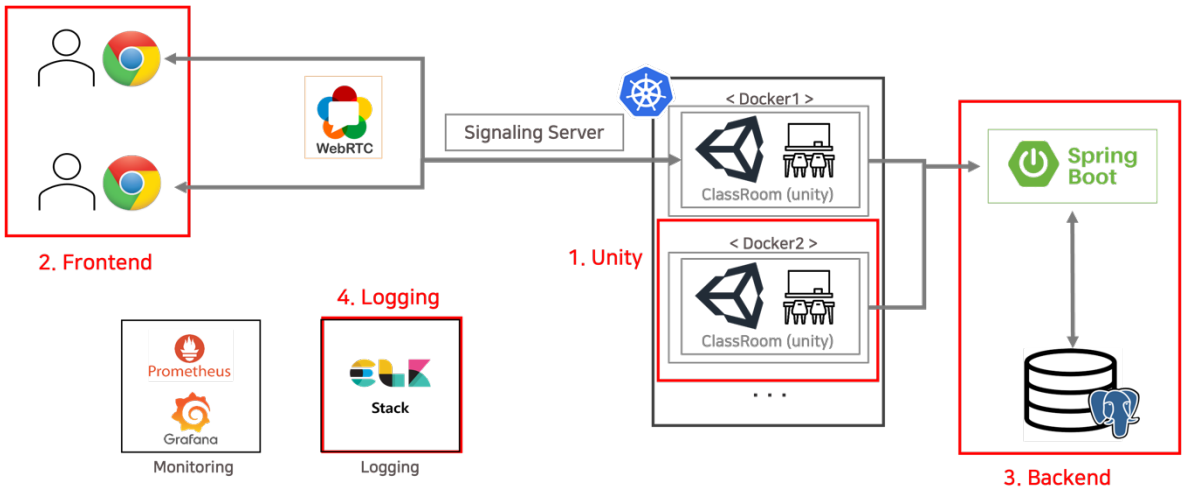

그림 1. 클라우드 게이밍 구조도

i. WebRTC

Unity와 사용자 브라우저 간 연결하기 위해 WebRTC 기술을 활용하여 연결한다. 이는 P2P 방식으로 연결되어지며 통신 과정은 다음과 같다.

1. Unity, 브라우저 간 P2P 통신에 동의

2. 서로의 주소를 공유

3. 방화벽 우회

4. 멀티미디어 데이터를 실시간 공유

Unity와 브라우저 간 데이터를 교환하기 위해서는 위 2번의 과정처럼 연결하려는 ip, port 정보를 알아야한다. 이는 클라이언트에서 해결할 수 없는 문제로 이를 통신 중재자의 역할로 시그널링(signaling) 서버를 구축하여 해결한다. 라우터의 public ip 주소만 찾는다 하여 WebRTC 연결을 할 수 없다. Public ip 정보뿐만 아니라 해당 네트워크에 연결된 private ip 주소까지 알아내야 특정 사용자를 지정할 수 있다. NAT(Network Address Translation)는 private ip 값을 public ip 값으로 대응시켜주는 장치이다. NAT의 도움으로 한 대의 라우터로도 여러 단말 기기들이 다른 서버와 통신할 수 있도록 한다. 두 클라이언트의 private ip 간에 통신이 가능하도록 하는 기술을 NAT Traversal이라 한다.

NAT Traversal의 대표적인 방법에는 직접 연결(Hole Punching) 방식의 STUN, 중계 연결(Relaying) 방식의 TURN 2가지 방법이 있다. STUN(Session Traversal Utilities for NAT)은 클라이언트가 STUN 서버에 요청을 보내어 자신의 ip, port를 반환하도록 한다. 라우터의 방화벽 정책(UDP 프로토콜 제한, 포트 제한)과 같은 이유로 STUN 서버를 통해 정보를 찾지 못하는 경우 TURN(Traversal Using Relay NAT) 서버를 이용한다. TURN 서버는 네트워크 미디어 패킷을 중개하여 릴레이(relay) 방식을 사용하여 통신을 유지한다.

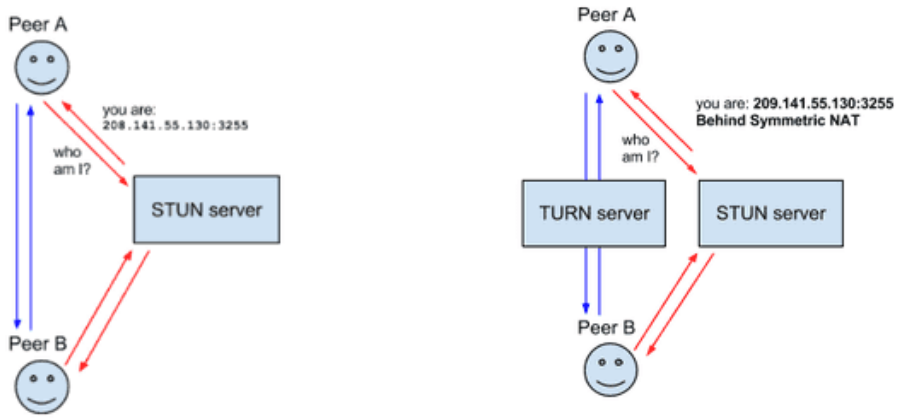

그림 2. STUN, TURN 서버의 통신 과정[1]

다양한 플랫폼, 환경에서도 안정적인 통신 유지를 위하여 먼저 STUN 서버를 통해 접속 시도 후 연결에 실패하면 TURN 서버를 통해 접속하도록 구성하였다.

[1] https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols

ii. Unity

사용자의 브라우저와 P2P 연결을 위해 Unity Redner Streaming API를 활용한다. ConnectionManager Class를 만들어 사용자가 게임에 입장할 때마다 독립적인 카메라를 가지도록 한다. 시그널링 서버와 통신할 프로토콜 규격을 정하고 연결할 후보들을 찾고 데이터를 주고 받는다.

WebRTC 기술로 사용자들과 연결하며 사용자들은 독립된 게임 플레이 화면을 제공한다. 마우스, 키보드, 터치 입력을 받을 수 있으며 그에 따른 상호작용을 주고 받는다.

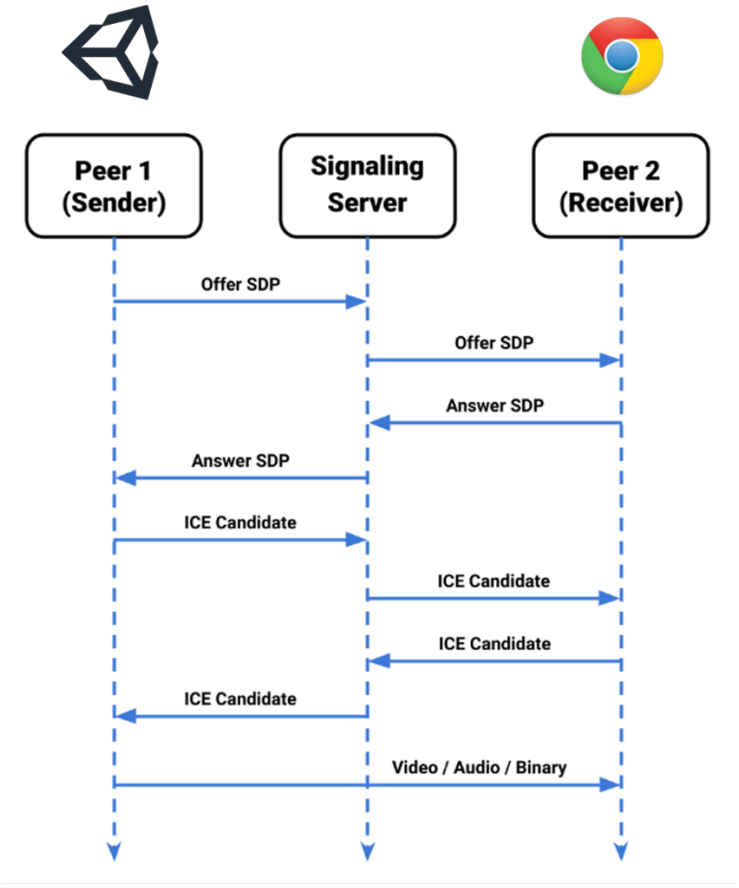

그림 3. Unity와 브라우저 간 WebRTC 통신 과정[2]

iii. Frontend

Unity와 P2P 연결을 위해 WebRTC API를 활용한다. 시그널링 서버와 통신할 프로토콜 규격을 정하고 연결할 후보들을 찾고 데이터를 주고 받는다. 프론트 페이지, I/O 입력, Unity와 데이터를 주고 받고 통신하기 위한 채널을 구현하였다.

[2] https://docs.unity3d.com/Packages/com.unity.renderstreaming@3.1/manual/overview.html

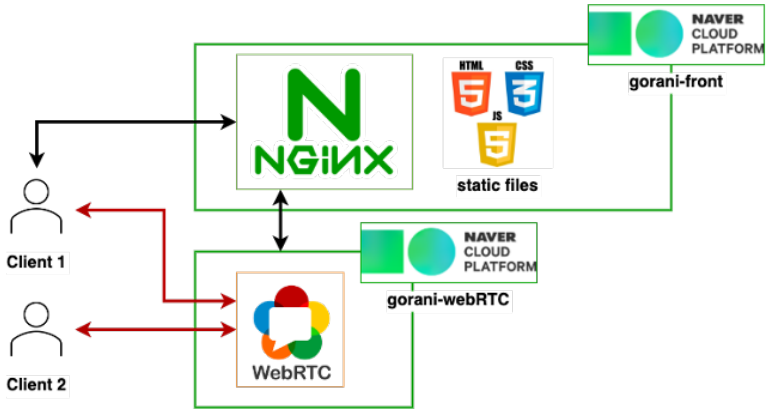

그림 4. Frontend 구조도

iv. Backend

- Spring Security를 이용한 JWT 방식의 인증/인가 구현
- Spring Data JAP & PostreSQL을 이용한 ORM 방식의 코드 설계
- Spring Rest Docs를 이용한 API 문서화

v. Logging

가벼운 애플리케이션을 위해 logstash 대신 filebeat로 로그 수집을 한다.수집한 로그 데이터를 logstash 서버로 보내 인덱싱을 진행한다. Elastic search과 kibana를 통해 시각화하여 로깅 시스템을 구축하였다.

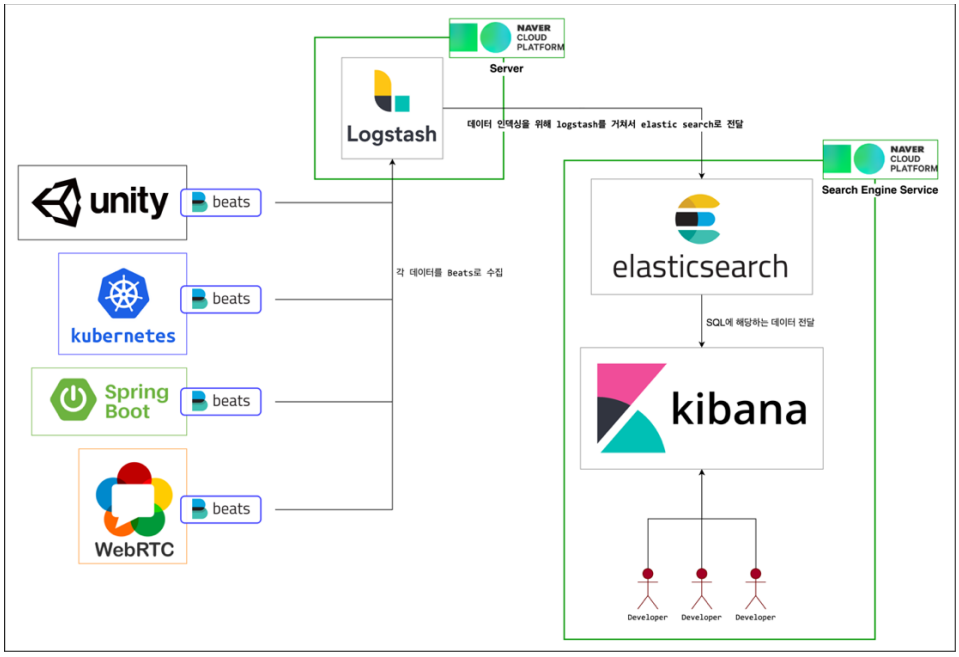

그림 5. 로깅 시스템 구조도

III. 갱신된 과제 추진 계획

기존 cloud platform에서 gpu가 탑재된 인스턴스를 대여하여 unity 서버를 가동하기로 하였으나 예산 문제로 개인 pc를 포트포워딩하여 unity 서버를 구축하기도 변경하였다.

TODO		8월				9월	
		1	2	3	4	1	2
Back	Spring Boot, k8s 연동						
Front	회원가입 및 로그인 화면 구현						
	교실접속(초대 코드, 캐릭터 커스텀) 화면 구현						
Unity	Backend API에 따라 DB정보 조회 관련 기능 구현						
	수업 진행(화면 공유, audio streaming) 관련 기능 구현						
	게시판 관련 기능 구현						
Infra	Front/Back, Postgres, 시그널링 서버 컨테이너 구축						
	유니티 k8s 구축						
	로깅 & 모니터링 시스템 구축(ELK, 그라파나, 프로메테우스)						
ETC	구축 플랫폼 성능 측정 및 비교						
	최종보고서 및 발표 준비						

IV. 구성원별 진척도

i. 역할 분담

이름	역할 분담
정현모	API gateway 서버(Spring Boot) 개발 - **인증 API 구축(완)** - **교실 API 구축(완)** - k8s API 연동 웹 서버 프록시 및 로드 밸런싱(Nginx) 개발 - **API gateway 구축(완)** - 로드 밸런싱 구축 로깅 시스템 구축(ELK) - **ELK 구축 및 연동(완)** - Logstash filter 적용
이재욱	Unity 및 WebRTC 개발 - **Unity HTTP utility 제작(완)** - **Eggcation URS 버전 구동(완)** 모니터링 구축(Prometheus, Grafana) - Prometheus + Grafana 구축 및 연동 DB, Redis 설계 - **회원 테이블 설계(완)** - **교실 테이블 설계(완)**
전설	Unity 및 WebRTC 개발 - **시그널링, TURN 서버 구축(완)** - **Eggcation URS 버전 구동(완)** 프론트 개발 - **프론트 페이지 URS 연동(완)** - 안정적인 WebRTC 접속을 위한 테스트 코드 작성 CICD (Github Actions) 구축 - lint 적용 컨테이너 구축(k8s) - Unity 교실 생성 및 관리
공통	Unity Render Streaming 기술 연구 - **WebRTC 연구(완)** - **URS 분석 후 코드 커스텀(완)** - **멀티 플레이 접속 테스트(완)** 아키텍처 설계 - **전체 아키텍처 설계(완)** 중간 보고서 작성

V. 보고 시점까지의 과제 수행 내용 및 중간 결과

프로그램 코드들은 GitHub로 관리되며, 각 기능에 따라 repository를 만들어 진행된다. 현재 server, front, unity, database, webrtc 5개의 repository로 관리하고 있다.

Unity Render Streaming API 적용과 WebRTC 통신 과정을 구현하여 프론트, WebRTC는 퍼블릭 클라우드(NCP)에 구축하고 unity는 그래픽 카드가 있는 개인 PC에 구축하여 테스트를 진행하였다. 서로 다른 두 클라이언트 브라우저에서 프론트 URL에 접속하면 플레이어마다 독립된 카메라가 제공되고 게임 플레이가 가능하다. 키보드(WASD)로 캐릭터를 움직일 수 있으며 마우스로 카메라를 조정할 수 있다.

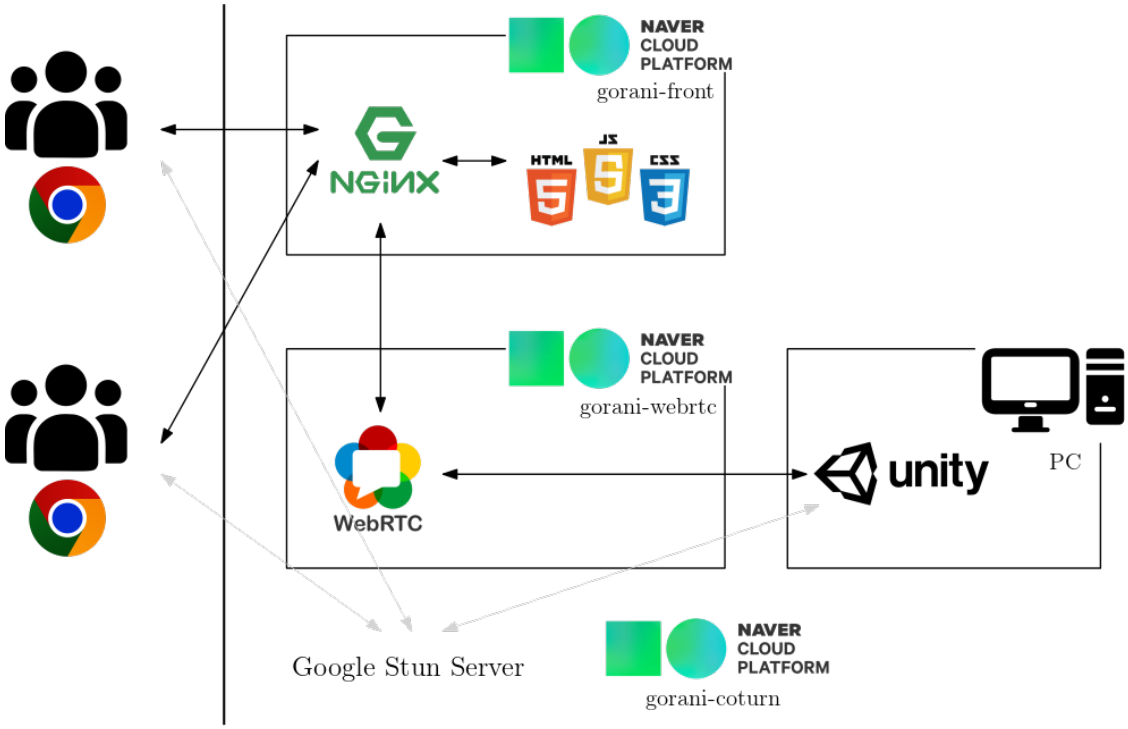

그림 6. Unity Render Streaming 테스트 환경 구조도

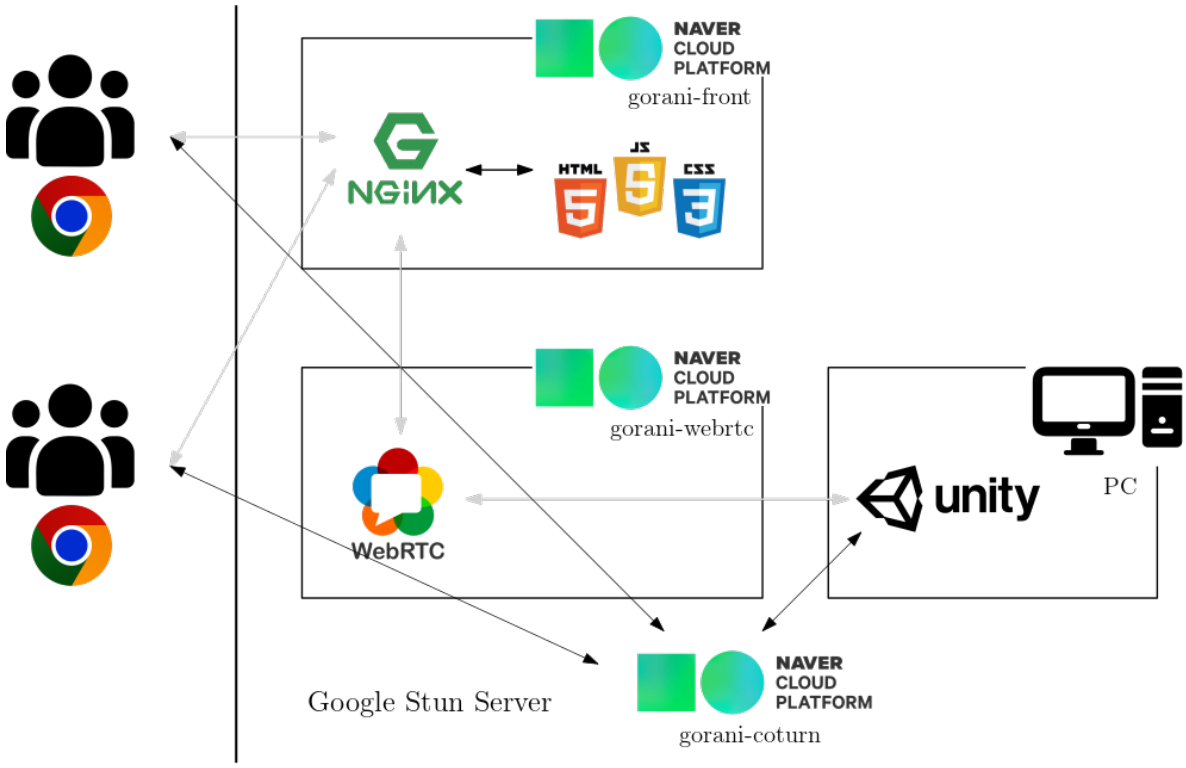

그림 7. TURN 서버를 통해 WebRTC 연결이 된 구조도 예시

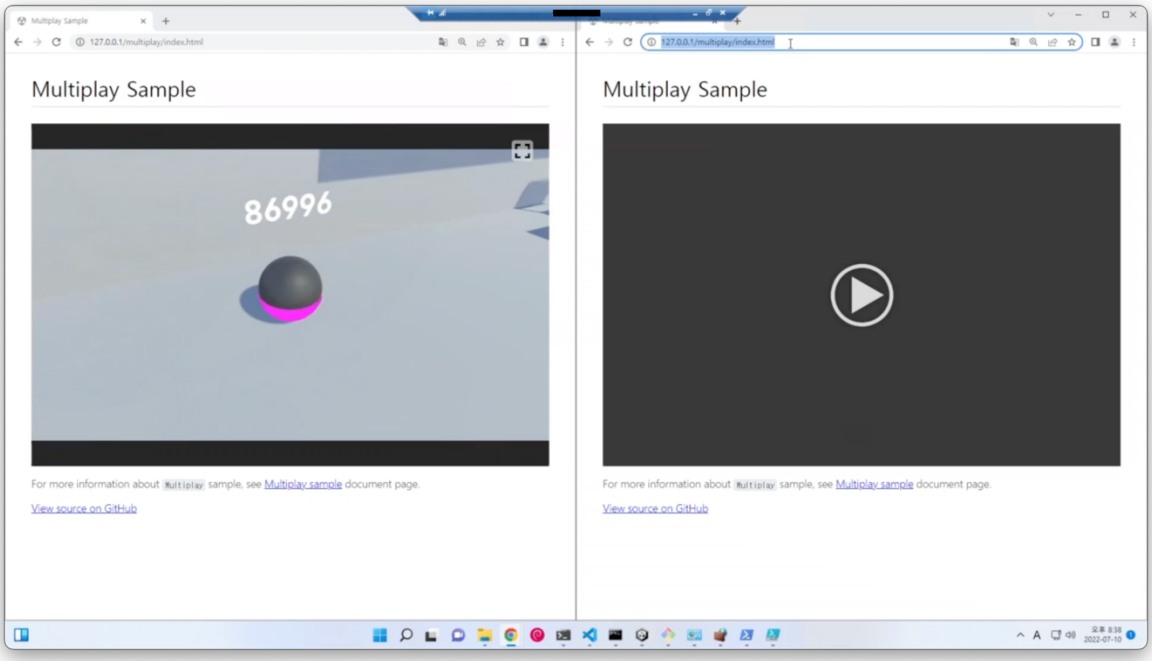

그림 8. Unity에서 게임을 실행한 모습

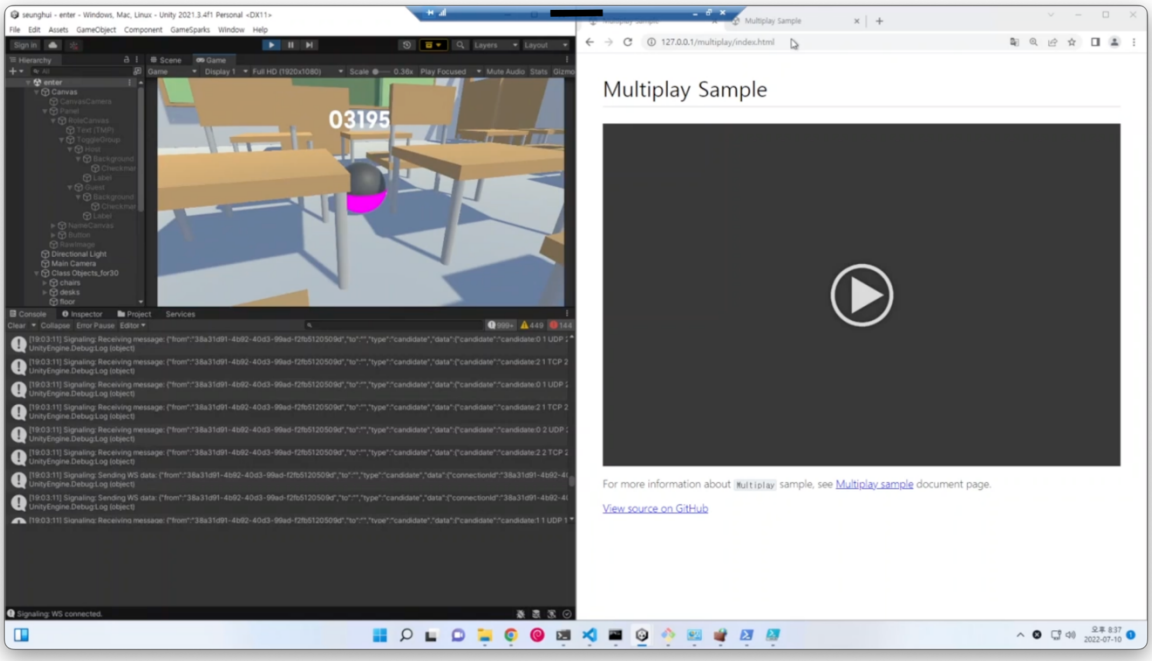

그림 9. 브라우저에서 WebRTC 통신하여 게임을 실행한 모습

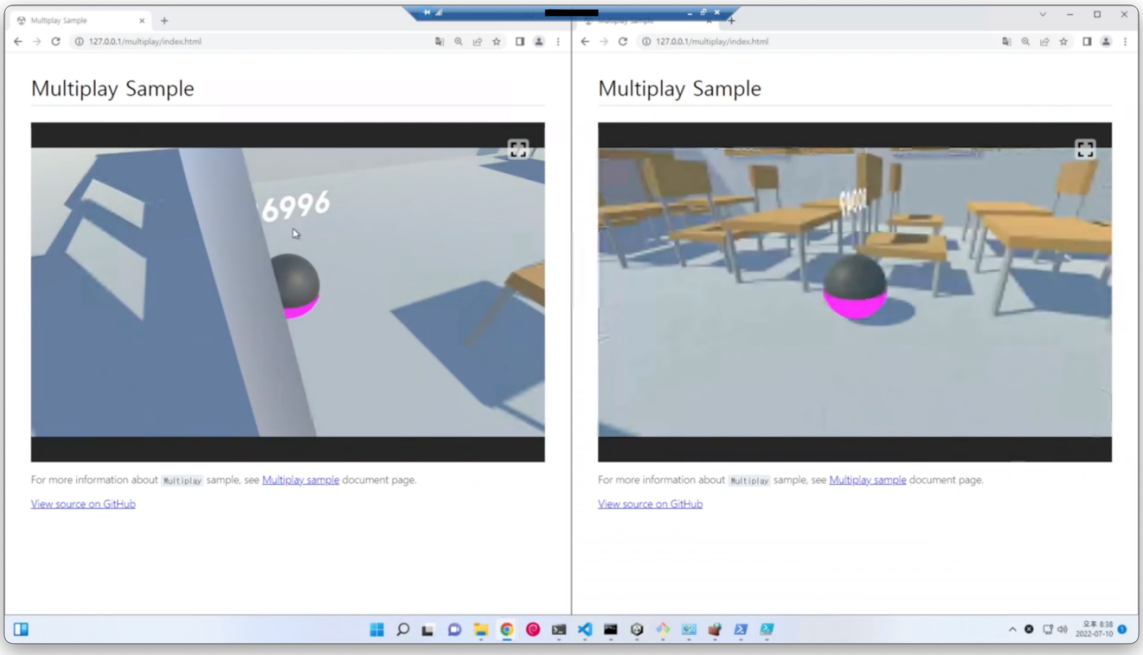

그림 10. 다른 브라우저에서 접속하여 게임을 실행한 모습

i. webrtc

- 시그널링 서버 코드를 모아둔 repository
- WebRTC 통신을 위한 SDP 교환, ICE candidate 교환 작업 처리
- prettier, lint 적용
- 테스트 코드 작성
- dockerize 진행(ncp cloud 구축)

ii. front

- 프론트 코드를 모아둔 repository
- 브라우저 호환을 위한 adapter, polyfill 적용
- WebRTC 통신을 위한 SDP 형태 설정 및 연결
- 입력 장치 인식 및 상호작용 구현
- prettier, lint 적용
- 테스트 코드 작성
- dockerize 진행(ncp cloud 구축)

iii. unity

- Unity 코드를 모아둔 repository
- Unity Render Streaming 코드 분석 및 적용
- 멀티플레이 상호작용
- 브라우저 데이터 통신을 위한 채널(DataChannelBase) 생성

iv. server

- Spring Boot 코드를 모아둔 repository
- JWT 기반 인증 API
- 교실 생성 API

v. database

- Postgres query 코드를 모아둔 repository

STUN 서버만으로 접속을 시도한 경우 라우터 방화벽 정책 혹은 브라우저 보안 정책으로 클라이언트 간 연결이 되지 않는 경우가 있었다. 사용자한테 일관적인 서비스 제공이 어려웠다.

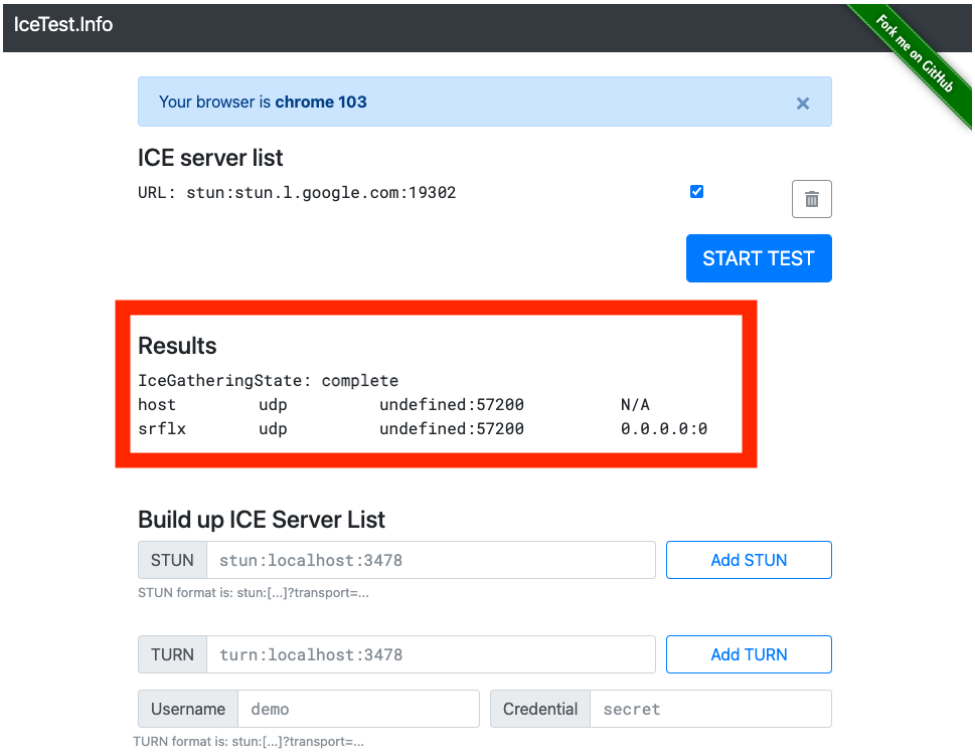

그림 11. STUN 서버를 통한 통신 연결 실패한 상황

이를 해결하고자 TURN 서버의 구축으로 안정적인 연결이 되었다. 하지만 통신을 할 때마다 TURN 서버를 경유하여 네트워크 환경에 따라 게임 플레이 환경에 영향을 끼쳤다.

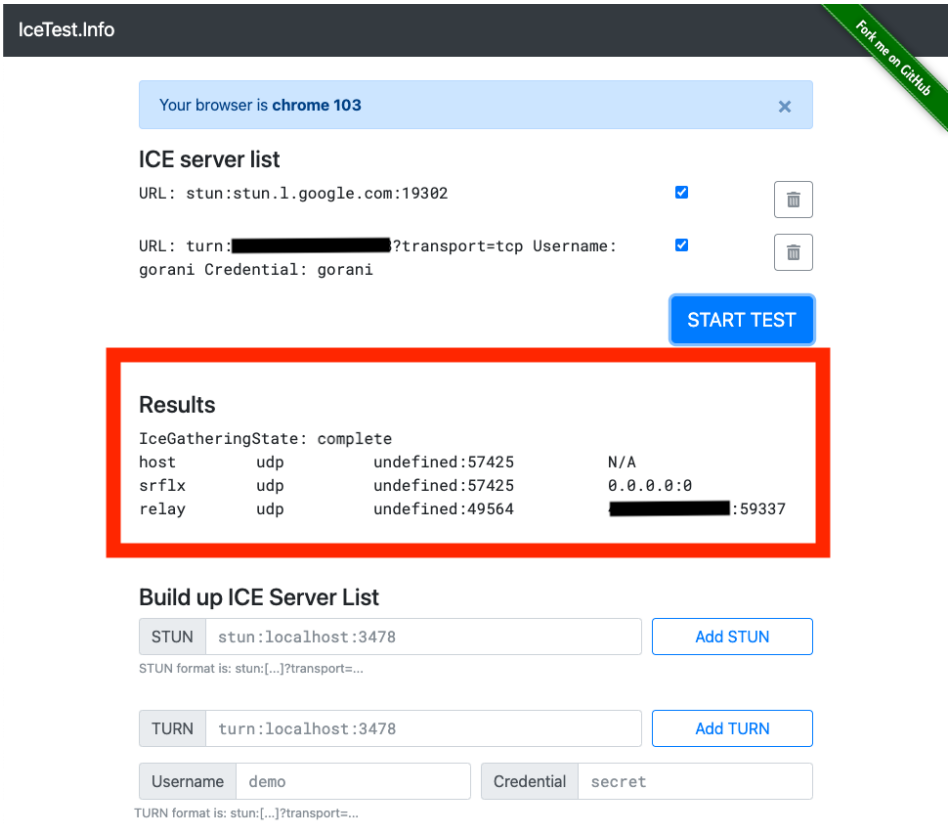

그림 12. TURN 서버의 추가로 통신 연결 성공이 된 상황

퍼블릭 클라우드를 통해 접속하는 것보다 같은 로컬 환경에서 연결하면 더욱 안정적인 게임 플레이를 할 수 있는 것을 확인하여 네트워크 통신 중 발생하는 overhead가 서비스 품질에 영향을 끼친다는 것을 알 수 있다. 보다 쾌적한 게임 플레이를 위해 WebRTC 통신 프로토콜에 대한 이해가 필요하고 Unity Render Streaming의 성능 결과를 보기위해 많은 랜더링 연산이 필요한 고사양 게임을 비교군으로 두어 클라이언트 부하에 차이가 있는지 비교할 예정이다.