# YamYam

NoPOKER 발표회

# 목차

# NoPOKER

인디언 포커 + 난투 액션

# 보드게임을 좋아하는 4명이 만났다..

심리전

반응속도

변하지 않았던 가치

# 뇌지컬과 피지컬을 필요로 하는 보드액션게임

# Phase 1. 인디언 포커

랜덤으로 카드를 뽑고, 다른 플레이어들의 카드와 배팅을 보고 본인의 배팅을 결정

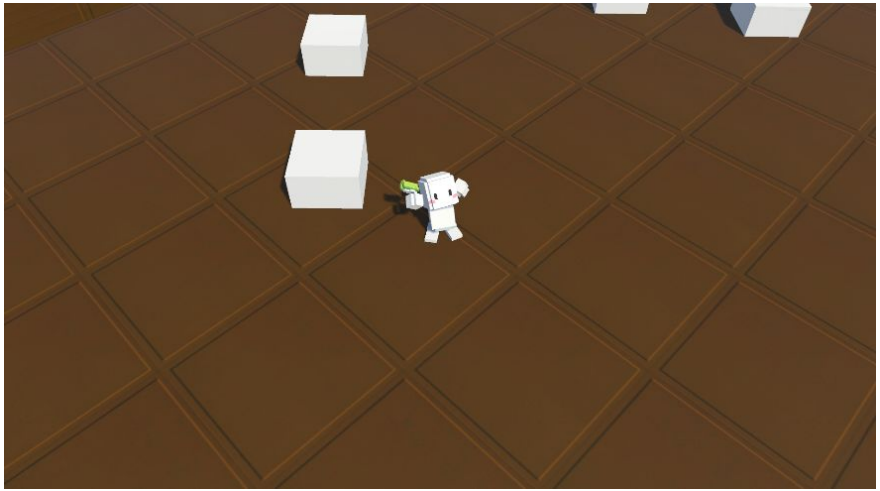
본인의 카드를 알 수 없기 때문에 심리전이 중요

# Phase 2. 난투

카드 숫자를 바탕으로 무기를 지급 받음 (1단계 ~ 10단계)

난투에서 살아남은 1인이 배팅된 칩을 모두 가져감

포커에서 안좋은 패를 받았더라도 피지컬로 극복 가능



# Experience

인증

인증 서버 구현
회원가입/로그인화면 구현

로비

로비 서버 구현
로비 구현

매칭

매칭 서버 구현
매칭 구현

2D
포커

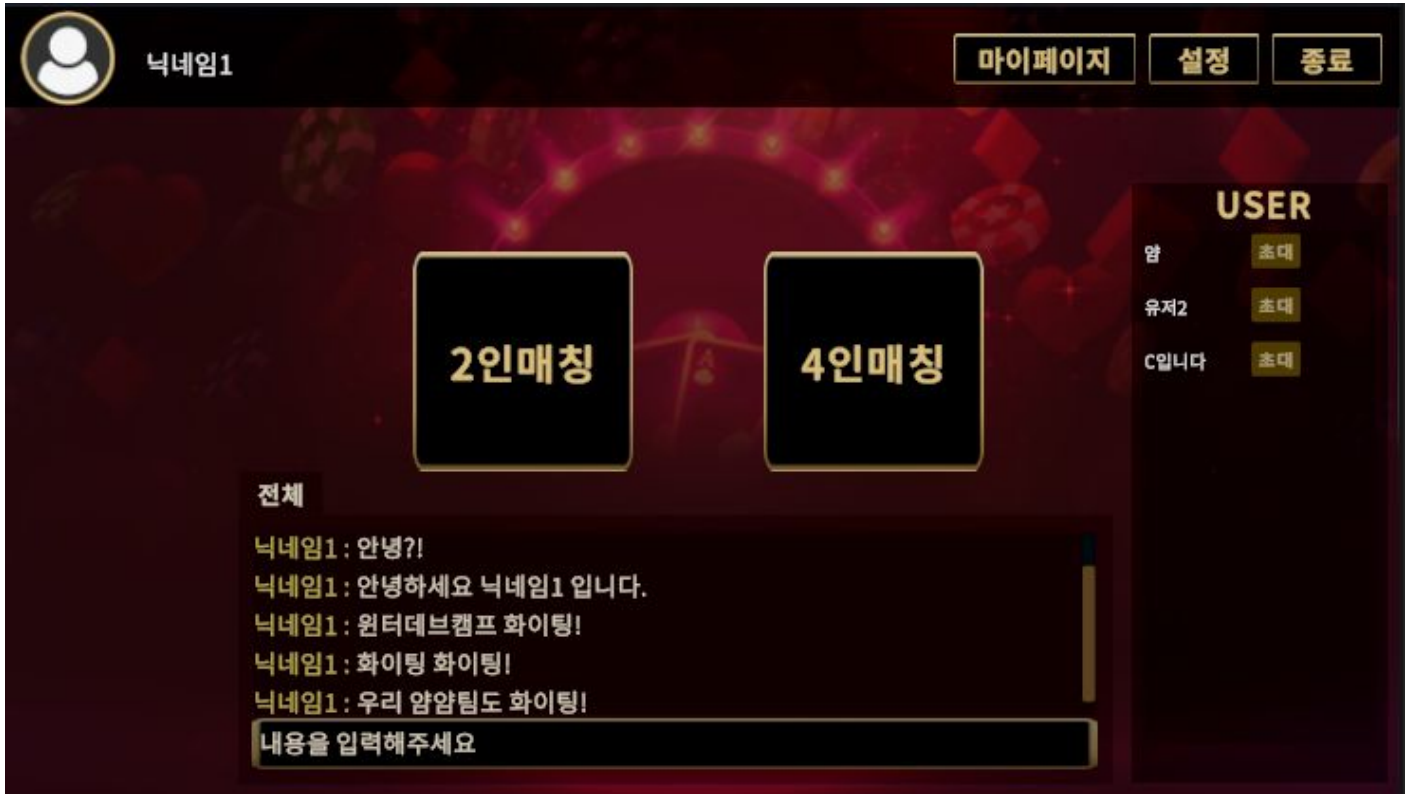
게임 서버 구현
2D 포커게임 구현

3D
액션

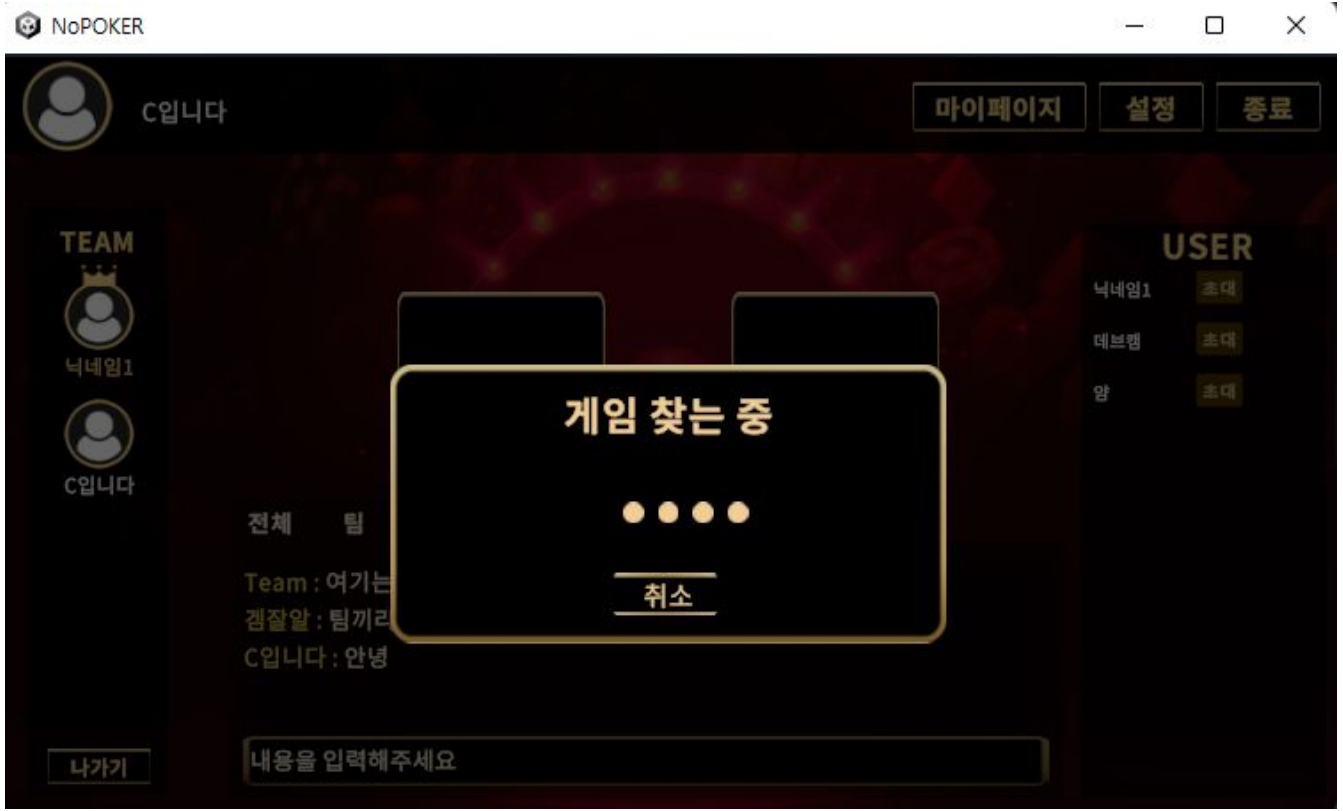
실시간 게임 서버 구현
3D 액션게임 구현

# Experience - 인증

# Experience - 로비



# Experience - 매칭



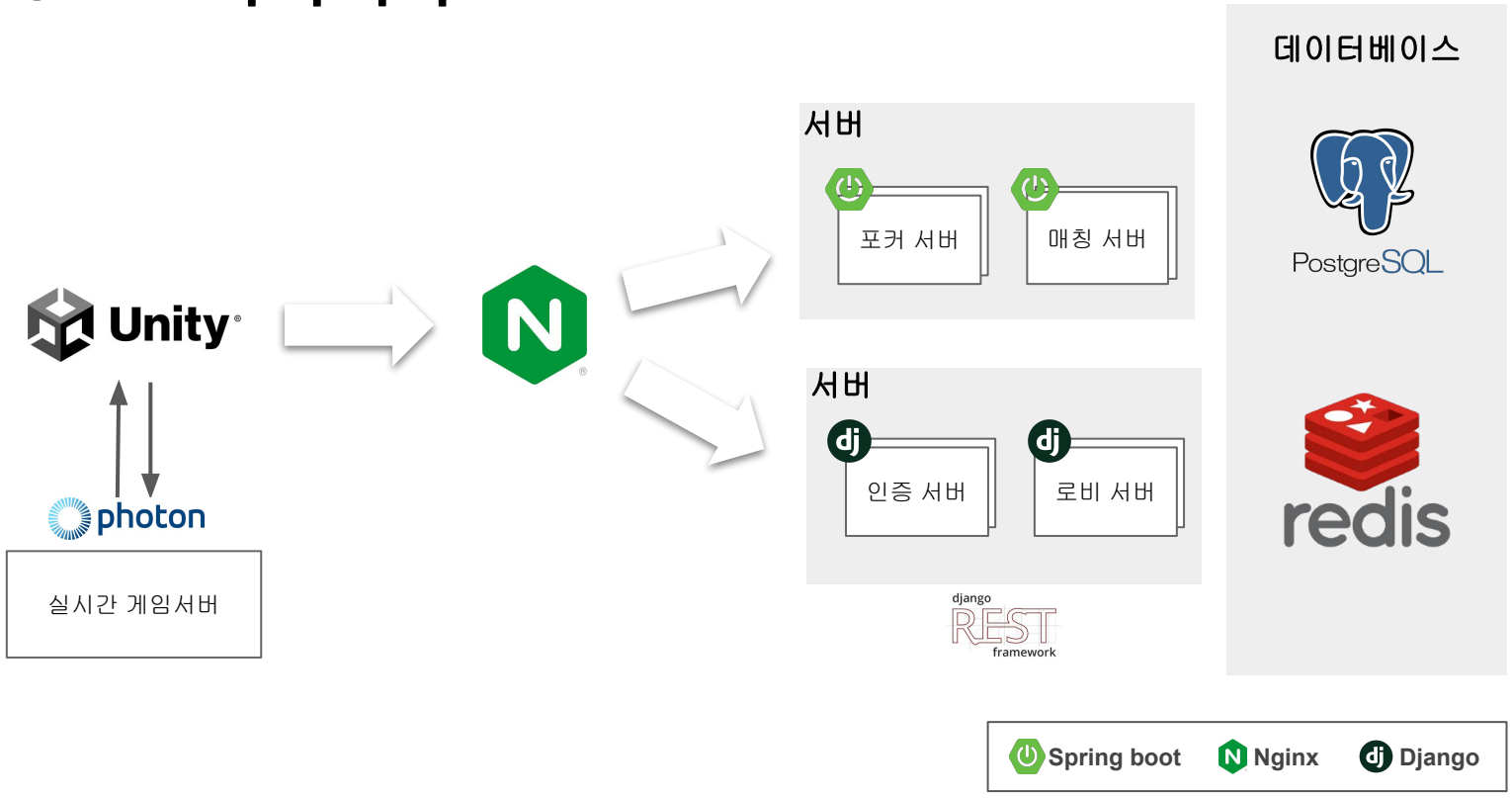
# Experience - 2D 포커

# Experience - 3D 액션

# 시연 동영상

# 아키텍처

# NoPOKER 아키텍처



# 개별 담당 부분 / 개인 회고

# API Gateway (Nginx) - 박동진

서비스 마다 인증 로직이 포함되는 것을 막기 위해 API Gateway 도입
Nginx에서 auth_request 모듈을 이용해 Token check를 수행하고 요청한 URL로 reverse proxy

인증이 필요한 API 리스트
- 마이 페이지
- 유저 정보 업데이트
- 회원 탈퇴
- 로비 웹 소켓 연결
- 매칭 웹 소켓 연결
- 게임 웹 소켓 연결

…

```
server {
    listen 80;
    error_log /var/log/nginx/example.error.log debug;
    large_client_header_buffers 4 16k;
    location /ws/lobby/ {
        if ($token_type = "") {
            return 400;
        }
        set $subqry $uri;
        set $clientip $remote_addr;
        proxy_pass_request_headers on;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        auth_request /check_token;
        proxy_intercept_errors on;
        error_page 401 = @error_401;
        proxy_pass http://lobby_server;
    }
```

# WebSocket - 박동진

클라이언트와 서버의 양방향 통신을 위해 WebSocket으로 로비 서비스 구현
- django channels를 이용해 connect, send, disconnect
- 초대 하는 유저(팀장)의 PK를 이용해 팀을 생성


DB index 1

Key : PK_NICKNAME
Value : 유저의 상태(팀장, 팀원, 게임 중)


DB index 2

Key : PK_NICKNAME (팀장)
Value : 팀장과 팀원의 정보가 담겨 있는 json

```
{
    "leader": {
        "id": 1,
        "nickname": "nickname1"
    },
    "invitees": [
        {
            "id": 2,
            "nickname": "nickname2"
        },
        {
            "id": 3,
            "nickname": "nickname3"
        }
    ]
}
```

# 시행 착오 - 박동진

## Access Token, Refresh Token

**첫번째 생각** : Access Token, Refresh Token을 같이 보냄
-> Refresh Token은 유효기간이 길기 때문에 탈취, 악용 위험이 큼

**두번째 생각** : Access Token을 먼저 보내고 401 이 리턴되면, Refresh Token API 호출
-> 클라이언트 단에서 refresh token api url을 알아야 함, 진정한 의미의 reverse proxy가 아님

**세번째 생각** : Access Token을 먼저 보내고 401 이 리턴되면, Refresh Token을 담고 동일한 api로 request
-> 클라이언트는 request를 최대 두 번 보내면 인증 여부를 알 수 있다.

```
map $http_token_type $token_type {
    default "";
    "Access" "access_token";
    "Refresh" "refresh_token";
}
```

```
location /check_token {
    internal;
    if ($token_type = "") {
        return 405;
    }

    proxy_pass_request_body off;
    proxy_set_header X-Original_URI $request_uri;
    proxy_set_header X-Api-Subqry $subqry;
    proxy_set_header X-Client-IP $clientip;
    proxy_set_header Host $host;

    proxy_pass http://auth_server/accounts/check_$token_type/;
}
```

# 시행 착오 - 박동진

## 마이크로 서비스 분리


기존 서비스 구조

추가 요구 사항
- 팀 초대
- 팀 매칭
- 팀 채팅

WebSocket connection을 최소화



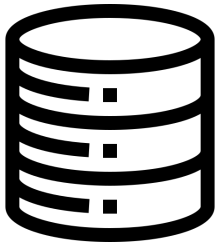
아키텍처의 Trade-Off 분석


변경된 서비스 구조

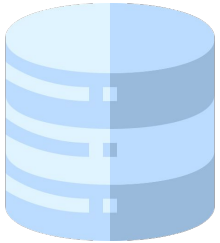
로비 서비스 -> 팀 초대, 팀 채팅
매칭 서비스 -> 팀 매칭

# 시행 착오 - 박동진

## 로비 서비스 DB 설계



**문제의 시작** : 인증 서비스의 유저 테이블을 참조하는 채팅 테이블 생성
-> 서비스 간의 데이터 전달을 위해 API 필요 -> 비효율적 -> 어떻게 해결하지?



**더 심각한 문제** : 로비 서비스에서만 사용하는 유저 테이블을 생성, 관리
-> 데이터 중복 문제
-> 인증 서비스의 유저 테이블과의 데이터 일관성 문제
-> 데이터를 수정할 때마다, 서로의 테이블에 비동기 태스크 큐로 API를 요청
-> 온라인 유저관리는 어떻게..? 팀 테이블은 또 어떻게 만들지..?
-> 이게 맞나..?

# 시행 착오 - 박동진

## 로비 서비스 DB 설계

고민하다 다시 본 요구 사항들 -> Database가 없어도 되겠는데?


온라인 유저 목록, 팀원 목록 -> Redis로 관리 가능 -> 유저 서비스와 의존성 X
과거 채팅 필요 없음 -> WebSocket이 연결된 상태에서 채팅을 출력



하나의 서비스가 무조건 Database를 가져야 한다는 고정관념이 있었음



중요한 것은 요구사항!
적절한 기술을 사용하자!

# 개인 회고 - 박동진

## 배운 점

- API Gateway에서의 인증
- WebSocket의 기본 구조
- docker로 개발, 배포 환경 세팅
- 문서화 (API 문서, 개인 공부)

## 어려웠던 점

- MSA를 고려한 설계
- Nginx 작동 방식 이해
- WebSocket 단위 테스트
- 배포를 고려한 환경 변수 세팅

## 아쉬운 점

- 성능 테스트를 못해본 것
- 클라이언트와 연동을 완벽하게 하지 못한 것
- 비즈니스 로직을 한 곳에 모으는 리팩토링을 하지 못함

# 개인회고 - 구자현

## 배운 점

- 기획은 한번에 확실히 정해도 부족하다. (수많은 수정..)

- 이론적으로만 알고 있던 네트워크 지식을 여러 툴들을 사용해 직접적으로 확인해보는 기회가 많아서 좋았다.

## 어려웠던 점

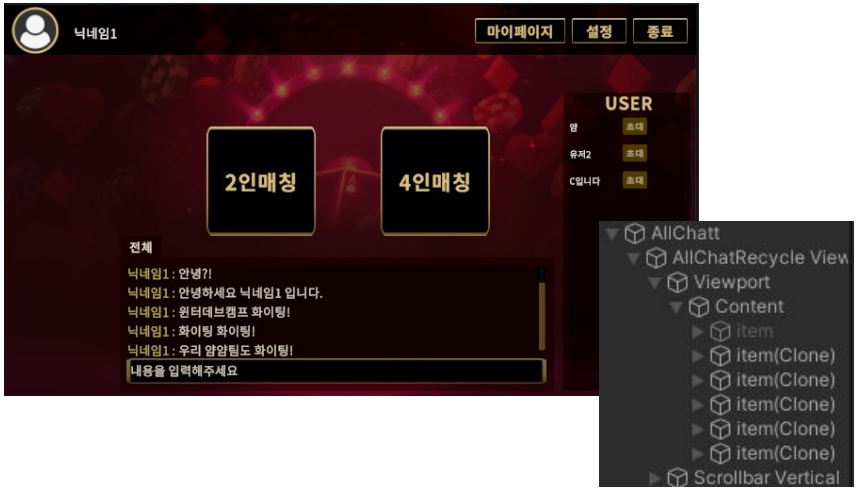
- STOMP spec이 Unity에서 잘 지원하지 않고 관련 레퍼런스도 없어 메시지 파싱에 대한 어려움이 있었음.
- 룰에 대한 헛점이 발견되어 계속해서 수정

## 아쉬운 점

- 연결에 시간이 많이 걸려 2차 마일스톤 구현을 못해서 아쉽다.
- 클라이언트의 입장을 생각해보지 않고 기술 스택을 (STOMP)선택한게 아쉬웠다.

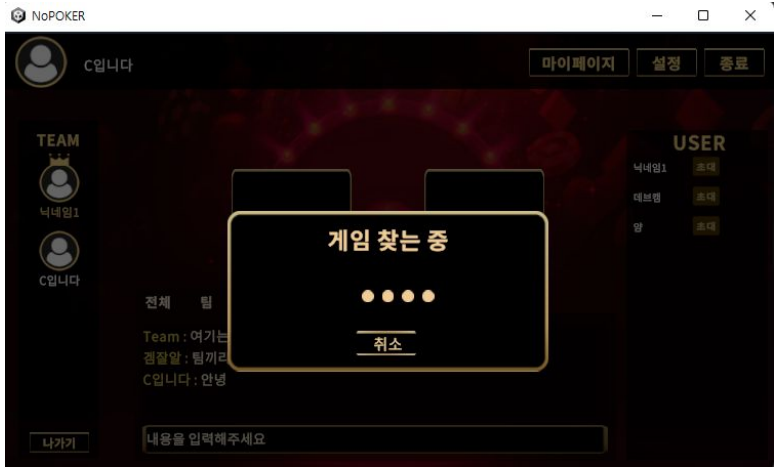
# 화면 구현 - 현명은

## 재사용 스크롤 뷰




채팅할때마다 생성되는 것이 아닌,
눈에 보이는것들만 보여준다

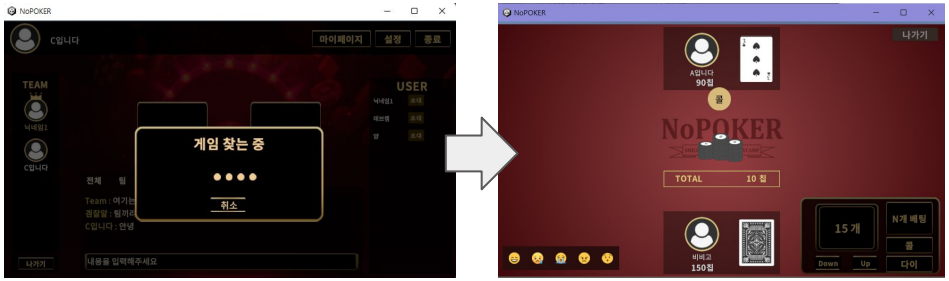
## 코루틴을 이용한 비동기처리




UI는 UI대로 처리
데이터는 데이터대로 처리

# 시행착오 - 현명은

## 유니티 메인쓰레드 제한




코드와 로직상으로 오류가 없었으나,
진행이 되지않는 문제 발생

## 서버통신을 위한 데이터


각 서버 마다 혹은 서비스마다 필요한
데이터들의 양식과 받는 양식이 다름

# 개인회고 - 현명은

## 배운 점

- 유니티에서의 비동기 처리
- 깃허브에 대한 지식 증가
- 모르는것을 물어보고,
모른다고 대답할 수 있는 용기
- MVC 모델 이해

## 어려웠던 점

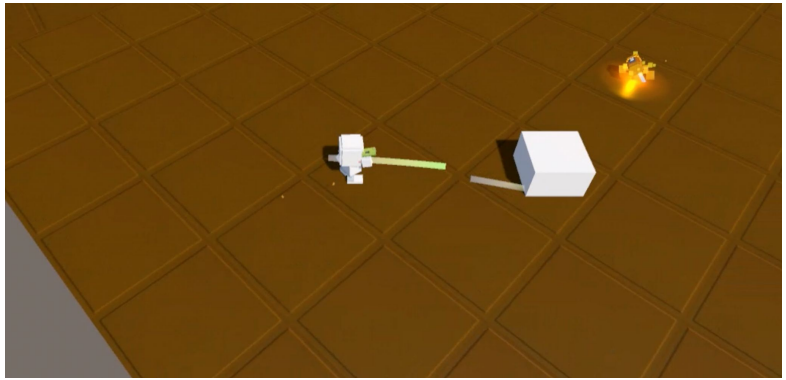
- 개인 테스트때 고려한
코드와는 다른 서버와의 연동
테스트에 필요한 코드

## 아쉬운 점

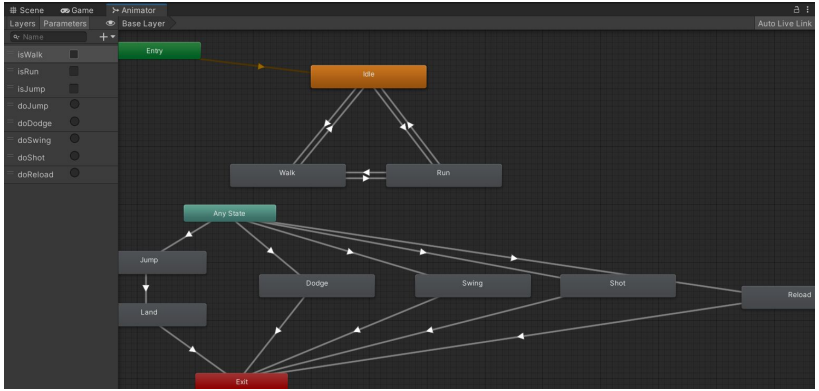
- 확장성을 고려하지 못한 설계
- 불완전한 게임
- 더티코드

# 3D 게임 구현 - 조현민

파티클 등을 포함한 그래픽



원거리 및 근거리 무기




3D 캐릭터 애니메이션

# 시행착오

- 피격 시 플레이어에게 피격으로 인식되지 않는 버그

  ➡ 게임 뷰에서 확인하는 것과 씬 뷰에서 확인할때 다르게 관측

- Input Manager등 유니티 자체적으로 저장되는 정보는 커밋하고 브랜치 체크아웃을 해도 그대로 가져가지기 때문에 경고가 뜨기도 했다.

- 포톤 적용하는 부분에 있어서 잘 적용이 안됨

  ➡ 강의영상 대신 공식 도큐먼트로 해결

# 개인회고 - 조현민

## 배운 점

- 시간이 남는다고 생각해도 막상 마감할때쯤 가면 부족하다.
- 그래픽과 애니메이션 적인 요소 구현 방법
- 포톤 열고 클라이언트가 방 입장, 퇴장 까지는 배움.

## 어려웠던 점

- 남은 작업량 체크할때 몇 퍼센트인지 계산하는게 어려웠다.
- 유니티와 C#에 대해 깊이가 얕았던 것 같다.

## 아쉬운 점

- 미완성으로 제출한다는 점이 아쉽다.
- 코드 작성에 있어서 안좋은 습관들을 못고친것 같다.

# 팀 목표

이슈를 조기 발견하여 이슈 해결비용을 줄이는 팀

## 이유

- 이슈를 나중에 고치려면 비용이 더 많이 듬
- 제한된 시간안에 효율적으로 일하기 위해
  데일리 스크럼을 통해 매일 진행 상황, 이슈를 공유

- 1/26 스크럼, gitignore 이슈
- 1/31 스크럼, 토큰 관련 이슈
- 2/1 스크럼, 비밀번호 찾기 이슈
- 2/14 스크럼, 데이터 전송 이슈

### 1월 26일 스크럼

| 이름 | 한 일 | 할 일 | 이슈 | 건강상태 | 질문 |
|---|---|---|---|---|---|
| 현명은 | - 회원가입 화면 구성<br>- 아이디및 비밀번호 찾기 화면구성 | - 브랜치 만들어서 PR하기<br>- 유니티 UI text 한글폰트 적용하기 | | 😊 | 화면더미 만들어놓은거 올릴려고 브랜치 하나 새로 만들려는데, client-login 이렇게 하면 될까요? |
| 조현민 | 플레이어 움직임 구현, 맵 구현 | 아이템 적용 | | 😊 | |
| 박동진 | User 어플리케이션 설계, django 어드민 페이지 구현 | 회원 가입 API 구현, 로그인 API 구현 | 이슈는 아니고 어드민 페이지를 리액트가 아닌 django 어드민 기능을 이용해서 구현하였습니다~~ | 😊 | @현명은<br>develop branch의 .gitignore.txt가 아니라 .gitignore로 해야하지 않나요??<br>헉...txt로 되어있군요 수정하겠습니다 |
| 구자현 | - 매치 서버 1차 데모 완성<br>- 게임 서버 1차 데모 프로젝트 구성 | - 매치 서버 알림 방식 변경 | | 🥶 | |

# 팀 회고

**지속적인 기획 변경**

기존 게임이 아닌 새로운 게임을 기획하다보니 기획자체에 허점이 많아 지속적인 기획변경이 있었음. 기획 변경을 하기에는 너무 시간이 부족한건 아니었을까 함

**연동을 미리 하지 않은 것**

연동을 미리 했다면 이슈를 빨리 발견해서 구현이 수월했을 듯
혼자 겪는 이슈가 아닌 같이 겪는 이슈가 훨씬 많다는 것을 깨달음

**개인위주의 느낌**

역할분담을 확실하게 하여 일을 진행하는데에 좋다고 생각하였으나,
서로의 역할에만 집중하여 개인위주의 느낌이 들어 아쉬움

# 팀 회고

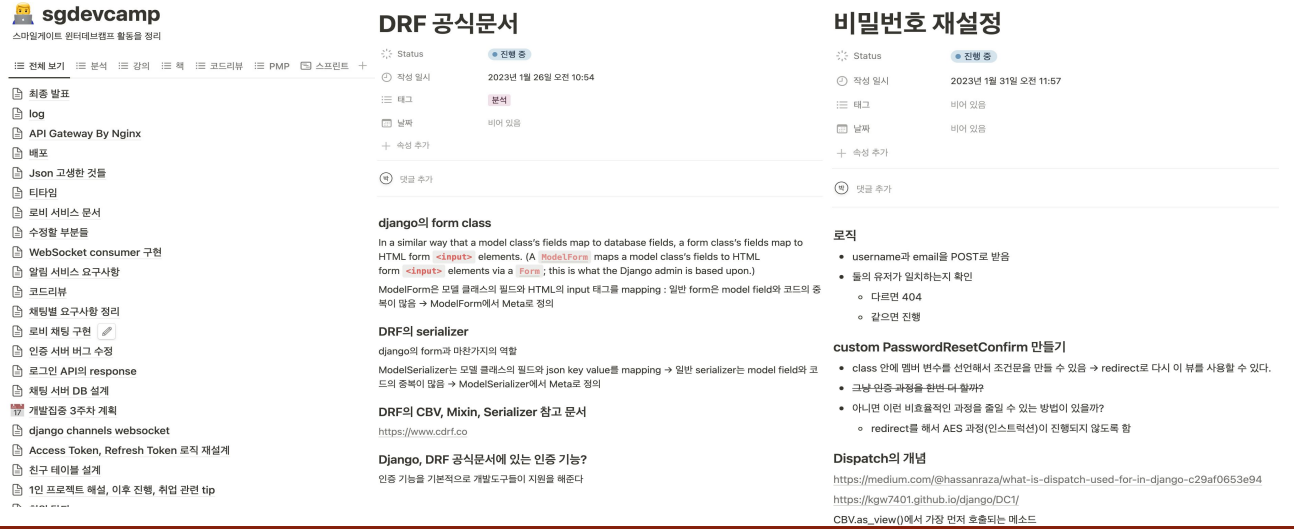
| | |
|---|---|
| **데일리 스크럼** | 스크럼을 통해 그날 해야할 일정을 체크하고, 리마인드할 수 있었으며 그날 작업에 동기부여를 주었음. |
| **다른 분야에 대한 이해도 증가** | 평소에 접하지 못하고 전혀알지 못하는 분야를 팀원과 소통하며 알게되고 접할 수 있게되어 내 분야가 아닌 다른 분야에 대해서도 어렵지 않게 접근이 가능함. |

# Appendix

# 개인 목표 - 박동진

## 1. 공식 레퍼런스를 보고 사용법을 스스로 익힐 수 있는 사람

notion에 공식 레퍼런스를 공부한 내용을 문서화 하고,
구현 중 생긴 이슈를 줄글로 논리적으로 작성하며 문제 해결의 실마리를 찾았다.



# 개인 목표 - 박동진

## 2. 단위 테스트 작성

테스트 코드를 작성하는데 시간이 오래 걸렸지만, 새로운 코드를 추가할 때 마다 직접 테스트를 하지 않아도 되는 경험을 해보며 단위 테스트 코드의 중요성을 깨달음

```
def test_create_duplicated_email_account(self):
    post = {"username": "user2", "nickname": "nickname2", "email": "user1@example.com",
            "password": "password2"}
    response = self.client.post(self.account_url, post)
    data = json.loads(response.content)
    self.assertEqual(response.status_code, 400)
    content = {"email": ['user with this email already exists.']}
    self.assertEqual(data, content)

def test_activate_account(self):
    response = self.client.get(self.get_activate_url)
    self.assertTemplateUsed(
        response, 'accounts/accounts_register_success.html')

def test_login_account(self):
    post = {"username": "user1", "password": "password1"}
    response = self.client.post(self.login_account_url, post)
    content = {"id": 1, "nickname": "nickname1"}
    self.assertEqual(response.status_code, 200)
    self.assertEqual(content, json.loads(response.content))
    self.assertEqual(response.has_header('Access-Token'), True)
    self.assertEqual(response.has_header('Refresh-Token'), True)
```

# 개인 목표 - 현명은

## 노포커 프로젝트에 있는 비동기코드를 이해하고 설명할 수 있는 사람

이전에 비동기를 다뤄본적이 없다가, 프로젝트를 통해 코루틴이 내가 알고있던 쓰임새뿐만이 아니라 여러가지 쓰임새가 있다는것을 알게되었다. 그리고 유니티에서는 메인쓰레드의 영역을 다른 쓰레드가 침범할 수 없어 이를 해결하기 위한 방법들을 직접 적용해보며 프로젝트에 쓰인 비동기 코드의 이해도를 높이고 설명할 수 있게되었다.

## 코드리뷰를 두려워하지 않는 개발자

누군가에게 코드 자체를 보여주는것이 부끄럽고 두려웠으나, 이번 프로젝트에서 계속 막히는 부분으로 인해 프로젝트 진행이 되지않는 상황이기에 팀원들에게 도움을 요청하였고, 새롭게 알게된 라이브러리 기능이 있을 경우에는 팀원들과 함께 공유하며 코드에 대해서 설명도 해보았다. 도움을 받고 설명을 하면서 내가 작성한 코드에 대해서 다같이 보고 얘기를 하였는데 부끄러웠으나 실질적인 도움을 많이 받았기에 앞으로의 코드리뷰는 전보다 두렵지 않을것이다.