Survival Game

하진혁

목차

게임 개요

Survival Game

장르: 2D 슈팅 로그라이트(뱀서라이크)
플랫폼: 안드로이드, IOS
사용 기술: Unity6, Node.js, MySQL
제작 기간: 4주
게임 주제: 몰려오는 몬스터들을 물리치며 레벨업하고
생존하는 게임

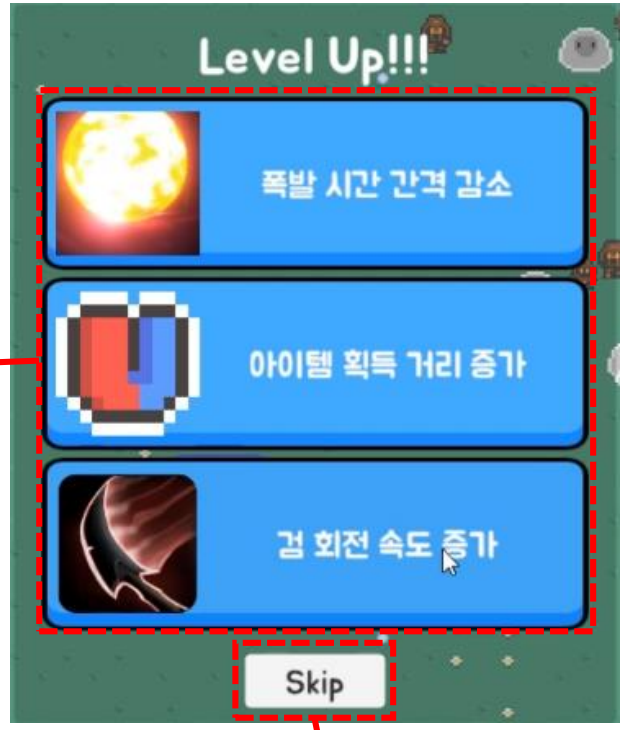
E-mail: hjhmoon61@naver.com
Github: https://github.com/HaJinhyeok/MiniGame.git
Youtube: https://youtu.be/NIE7aal20YE

게임 진행 _ 로그인, 회원가입

게임 종료

게임 진행_게임화면

게임 진행 _ Level Up



특전 선택

특전 선택 스킵

최대 15level

레벨업 할 때마다 특전 선택을 통한 업그레이드 가능

17가지 특전 존재

50level 달성 시 클리어

게임 진행 _ Wave

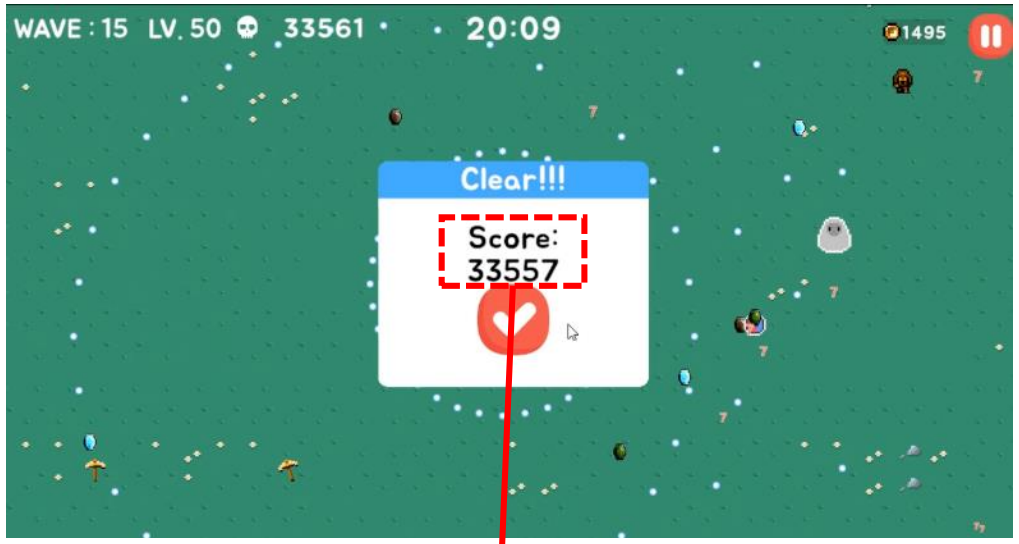
최대 15Wave

다섯 단계마다 골렘 웨이브 등장

최종 15Wave에서 생존하여
16Wave에 도달하면 클리어

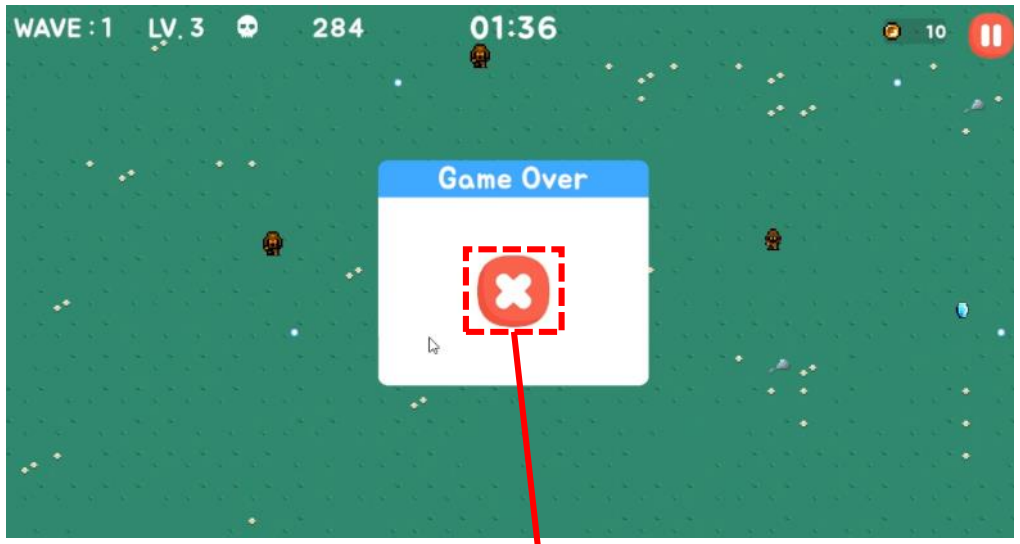
게임 진행

게임 클리어 화면



클리어 스코어

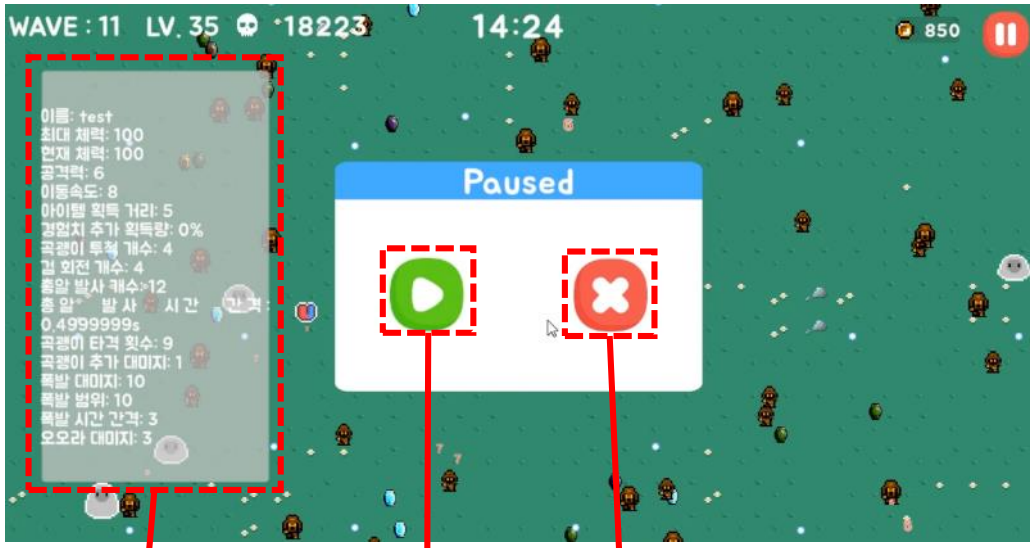
게임 오버 화면



결과 화면으로

게임 진행

일시정지 화면



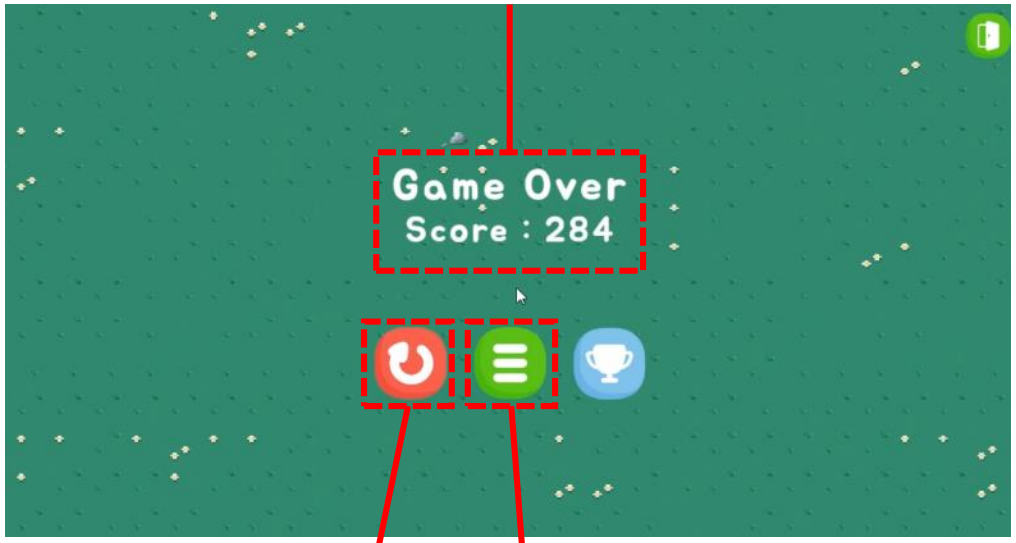
스테이터스 창

일시정지 해제

게임 중단

결과 화면

종료 스코어



재시작

시작 화면으로

구조 및 설계_ 아이템

경험치

레벨	이미지	경험치 획득량
Lv. 1		3
Lv. 2		6
Lv. 3		10
Lv. 4		20
Lv. 5		30

구조 및 설계_ 아이템

이름	이미지	효과
코인		인게임 재화 5 획득
빵		플레이어 체력 20 회복
자석		넓은 범위 내의 경험치와 코인 아이템 끌어 당기기

구조 및 설계_무기

이름	이미지	효과
총알		탄막형으로 자동 발사되는 기본 공격
검		플레이어 주위를 원형으로 돌며 닿는 적에게 대미지
곡괭이		일정 시간 간격으로 던져지며 닿는 적에게 대미지 타격 횟수, 추가 대미지
오오라		플레이어에 가까이 접근하는 적에게 틱 대미지
폭발		일정 시간 간격으로 폭발하며 범위 내 적에게 대미지

구조 및 설계_무기

```
void GetShot()
{
    Vector2 direction;
    int shotNum = GameManager.Instance.ShotInfo.ShotNum;
    float angle = 360f / shotNum;

    // pool에서 비활성화인 object 찾아서 우선 활성화
    for (int i = 0; i < ShotList.Count; i++)
    {
        if (!ShotList[i].activeSelf)
        {
            direction = new Vector2(Mathf.Cos(shotNum * angle * Mathf.Deg2Rad), Mathf.Sin(shotNum-- * angle * Mathf.Deg2Rad));
            ShotList[i].SetActive(true);
            ShotList[i].transform.position = transform.position;
            ShotList[i].GetComponent<ShotController>().SetDirection(direction.normalized);
        }
    }
    // 남은 개수는 새로 생성
    for (int i = 0; i < shotNum;)
    {
        direction = new Vector2(Mathf.Cos(shotNum * angle * Mathf.Deg2Rad), Mathf.Sin(shotNum-- * angle * Mathf.Deg2Rad));
        GameObject shot = Instantiate(Shot);
        shot.transform.parent = _shotPool.transform;
        shot.transform.position = transform.position;
        shot.GetComponent<ShotController>().SetDirection(direction.normalized);
        ShotList.Add(shot);
    }
}
```

Obejct Pool을 활용한 총알 관리

```
IEnumerator CoThrowPickaxe()
{
    while (true)
    {
        yield return new WaitForSeconds(3f);
        for (int i = 0; i < GameManager.Instance.PlayerInfo.AxeNum; i++)
        {
            PickaxeController pickaxeController = ObjectManager.Instance.Spawn<PickaxeController>(transform.position);

            pickaxeController.SetAngle(i);
        }
    }
}

참조 1개
public void StartExplosion()
{
    _isExplosionActivated = true;
    StartCoroutine(CoInvokeExplosion());
}

참조 2개
IEnumerator CoInvokeExplosion()
{
    while (_isExplosionActivated)
    {
        yield return new WaitForSeconds(GameManager.Instance.WeaponInfo.ExplosionInterval);
        Explosion();
        AudioManager.Instance.ExplosionSound.Play();
        ObjectManager.Instance.ExplosionEffect();
    }
}
```

Coroutine을 활용하여 일정 시간 간격으로 곡괭이 투척 및 폭발 이펙트 발생

구조 및 설계_무기

```
int currentSwordNum = GameManager.Instance.PlayerInfo.SwordNum;
SwordController swordController;

if (currentSwordNum >= 4)
    return null;
if (_pooledObject.ContainsKey(type))
{
    // 맨 앞 소드의 현재 각도
    float angle = _pooledObject[type][0].GetComponent<SwordController>().Angle;
    float angleDiff = 360f / (currentSwordNum + 1);

    for (int i = 0; i <= currentSwordNum; i++)
    {
        if (i == currentSwordNum)
        {
            swordController = ObjectManager.Instance.Spawn<SwordController>(pos);
            swordController.transform.parent = _parentObject[type].transform;
            swordController.Angle = (angle + i * angleDiff) % 360f;
            _pooledObject[type].Add(swordController.gameObject);
        }
        else
        {
            // 기존 소드들 각도 조정
            swordController = _pooledObject[type][i].GetComponent<SwordController>();
            swordController.Angle = (angle + i * angleDiff) % 360f;
        }
    }

    return _pooledObject[type][currentSwordNum] as T;
}
// 첫 소드 생성할 때
else
{
    if (!_parentObject.ContainsKey(type))
    {
        GameObject go = new GameObject(type.Name);
        _parentObject.Add(type, go);
    }
    SwordController obj = ObjectManager.Instance.Spawn<SwordController>(pos);
    obj.transform.parent = _parentObject[type].transform;
    List<GameObject> newList = new List<GameObject>();
    newList.Add(obj.gameObject);
    _pooledObject.Add(type, newList);
    return obj as T;
}
```

Object Pool 활용
일정한 간격으로 플레이어
주위를 회전하는 검

```
void Update()
{
    transform.position = ObjectManager.Instance.Player.transform.position;
    float offset = _transform.rotation.eulerAngles.z;
    offset += Time.deltaTime * 100;
    _transform.rotation = Quaternion.Euler(0, 0, offset);
    _coolTime += Time.deltaTime;
    if (_coolTime >= _interval)
    {
        _coolTime = 0f;
        AuraAttack(ObjectManager.Instance.Player.transform.position);
    }
}

참조 1개
void AuraAttack(Vector3 center)
{
    HashSet<EnemyController> enemyControllers = new HashSet<EnemyController>(ObjectManager.Instance.Enemies);
    foreach (var enemy in enemyControllers)
    {
        if (Vector3.Distance(center, enemy.transform.position) <= GameManager.Instance.WeaponInfo.AuraRadius
            && enemy.gameObject.activeSelf)
        {
            enemy.GetDamage(GameManager.Instance.WeaponInfo.AuraAtk, ObjectManager.Instance.Player.gameObject);
        }
    }
}
```

Object Pool 활용
근거리 적에게 일정 시간마다
대미지 부여하는 오오라

구조 및 설계_ 플레이어

조이스틱 조작으로 이동
몬스터와 충돌 시 출혈 및 붉어지는 이펙트(red flash)
일정 범위 내의 경험치와 코인 아이템 끌어당기는 패시브

```
참조 2개
public void PullItems(GameObject origin, float distance, float speed)
{
    foreach (var item in _pooledObject)
    {
        if (item.Key.Equals(typeof(Coin)) || item.Key.Equals(typeof(Exp_Lv1)) ||
            item.Key.Equals(typeof(Exp_Lv2)) || item.Key.Equals(typeof(Exp_Lv3)) ||
            item.Key.Equals(typeof(Exp_Lv4)) || item.Key.Equals(typeof(Exp_Lv5)))
        {
            foreach (var item2 in item.Value)
            {
                Vector2 direction = origin.transform.position - item2.transform.position;
                if (direction.sqrMagnitude < distance)
                {
                    item2.transform.Translate(direction.normalized * speed * Time.deltaTime);
                }
            }
        }
    }
}
```

자석 패시브

```
참조 2개
IEnumerator Blink()
{
    _currentColor.g -= 0.2f * _blinkFlag;

    if (_currentColor.g <= 0)
    {
        _currentColor.g = 0;
        _spriteRenderer.color = _currentColor;
        _blinkFlag = -1;
    }

    if (_currentColor.g >= 1)
    {
        _currentColor.g = 1;
        _spriteRenderer.color = _currentColor;
        _blinkFlag = 1;
        yield break;
    }

    yield return new WaitForSeconds(0.05f);
    StartCoroutine(Blink());
}
```

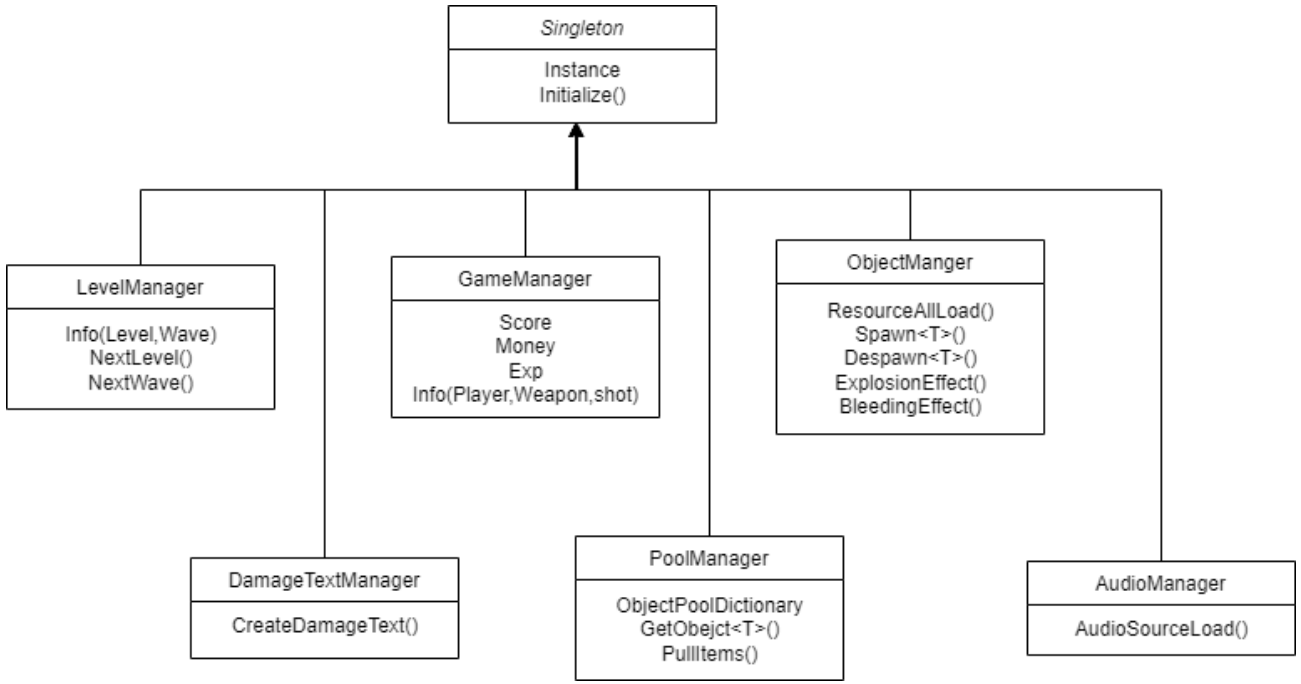
코루틴으로 구현한
플레이어 깜빡임(Blink) 이펙트

구조 및 설계_ 몬스터

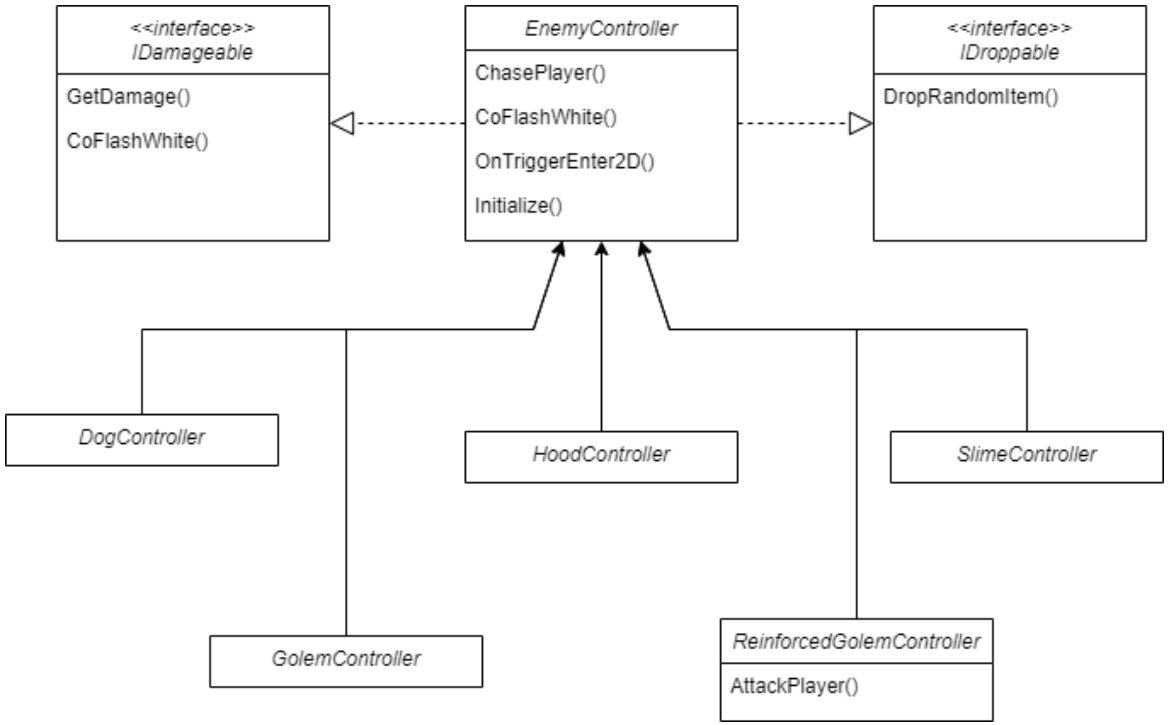
이름	이미지	스피드	공격력	체력
성체좀비		4	2	4
아기좀비		5	1	3
슬라임		3	3	7
골렘		2	5	15
강화된 골렘		2	10	30, 50

구조 및 설계

싱글톤 구조도



몬스터 구조도



구조 및 설계_ UnityWebRequest

UnityWebRequest의 HTTP요청 기능 중 POST를 활용해 JSON 형식 데이터(아이디, 비밀번호, 스코어) 를 서버로 송수신

```
참조 3개
private IEnumerator CoPost(string url, string json, System.Action<string> callback)
{
    var webRequest = new UnityWebRequest(url, "POST");
    var bodyRaw = Encoding.UTF8.GetBytes(json);

    webRequest.uploadHandler = new UploadHandlerRaw(bodyRaw);
    webRequest.downloadHandler = new DownloadHandlerBuffer();
    webRequest.SetRequestHeader("Content-Type", "application/json");

    yield return webRequest.SendWebRequest();

    if (webRequest.result == UnityWebRequest.Result.ConnectionError || webRequest.result == UnityWebRequest.Result.ProtocolError)
    {
        Debug.Log(webRequest.result);
        Debug.Log("네트워크 환경이 안좋아서 통신을 할 수 없습니다.");
    }
    else
    {
        callback(webRequest.downloadHandler.text);
    }
}
```

```
참조 1개
public void PostLoginData()
{
    Debug.Log("PostLogin");
    var url = string.Format($"{_host}:{port}/{_loginUri}");

    var req = new Protocols.Packets.req_login();
    req.cmd = 1000;
    req.id = GameManager.Instance.PlayerInfo.PlayerID;
    req.password = GameManager.Instance.PlayerInfo.PlayerPassword;

    var json = JsonConvert.SerializeObject(req);

    StartCoroutine(this.CoPost(url, json, (raw) =>
    {
        Protocols.Packets.res_message res = JsonConvert.DeserializeObject<Protocols.Packets.res_message>(raw);
        Debug.Log($"cmd: {res.cmd}, message: {res.message}");
        switch (res.cmd)
        {
            case 1201:
                // 아이디 없을 경우
                UI_PopUp.PopUpAction(Define.Warning_Inappropriate_PlayerID);
                break;

            case 1202:
                // 아이디는 존재하나 비밀번호가 틀린 경우
                UI_PopUp.PopUpAction(Define.Warning_Inappropriate_PlayerPassword);
                break;

            case 1203:
                // 로그인 성공
                UnityEngine.SceneManagement.SceneManager.LoadScene(Define.SurvGameScene);
                break;

            default:
                break;
        }
    }));
}
```

```
참조 1개
public void PostRegisterData()
{
    Debug.Log("PostRegister");
    var url = string.Format($"{_host}:{port}/{_registerUri}");

    var req = new Protocols.Packets.req_login();
    req.cmd = 1000;
    req.id = GameManager.Instance.PlayerInfo.PlayerID;
    req.password = GameManager.Instance.PlayerInfo.PlayerPassword;

    var json = JsonConvert.SerializeObject(req);

    StartCoroutine(this.CoPost(url, json, (raw) =>
    {
        Protocols.Packets.res_message res = JsonConvert.DeserializeObject<Protocols.Packets.res_message>(raw);
        Debug.Log($"cmd: {res.cmd}, message: {res.message}");
        switch (res.cmd)
        {
            case 1301:
                // 신규 유저 등록
                UI_PopUp.PopUpAction("등록 완료");
                break;

            case 1302:
                // 이미 존재하는 유저
                UI_PopUp.PopUpAction(Define.Warning_Already_Exist_ID);
                break;

            default:
                break;
        }
    }));
}
```

구조 및 설계_ 백엔드

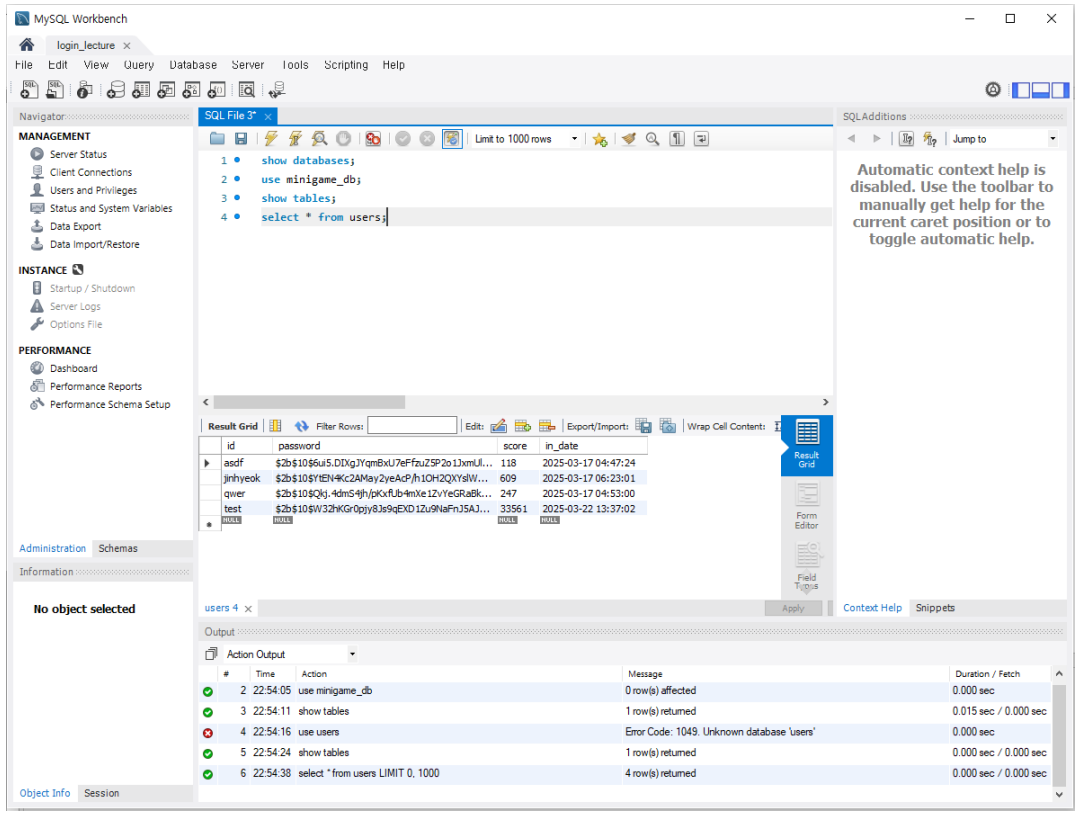
```
app.post("/login", async (req, res) => {
    let result = {
        cmd: -1,
        message: "",
    };
    let { id, password } = req.body;
    // select 쿼리문으로 정보를 가져와서 조회한다.
    const query = "SELECT * FROM users where id=?;";
    db.query(query, [id], (err, data) => {
        if (err) {
            console.log(`login error: ${err}`);
        }
        if (data[0] === undefined) {
            // 1. id가 없을 경우 - cmd 1201
            result.cmd = 1201;
            result.message = "ID가 없는데요";
        } else if (data[0].id === id && !bcrypt.compareSync(password, data[0].password)) {
            // 2. id는 존재하지만 password가 틀렸을 경우 - cmd 1202
            result.cmd = 1202;
            result.message = "비번 틀렸는데요";
        } else {
            // 3. id가 존재하고 password 또한 일치할 경우 - cmd 1203
            result.cmd = 1203;
            result.message = "로그인 성공";
        }
        res.send(result);
    });
})
```

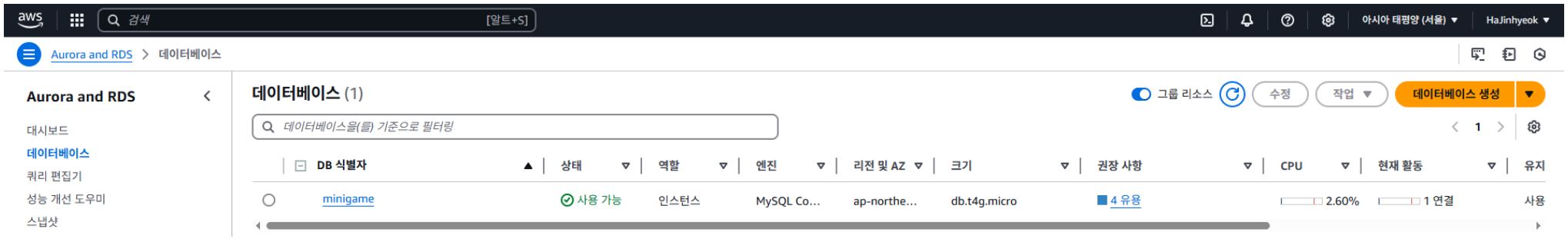
```
app.post("/register", async (req, res) => {
    let result = {
        cmd: -1,
        message: "",
    }
    let { id, password } = req.body;
    password = await hashedPassword(password);
    const query = "SELECT * FROM users where id=?;";
    const initScore = 0;
    db.query(query, [id], (err, data) => {
        if (data[0] === undefined) {
            // 1. id가 없는 신규 유저가 맞을 경우 - cmd 1301
            // 신규 유저 등록하기
            const registerQuery = "INSERT INTO users(id,password,score) VALUES(?,?,?);";
            db.query(registerQuery, [id, password, initScore], (err) => {
                if (err) console.log(`register error: ${err}`);
            })
            result.cmd = 1301;
            result.message = "신규 유저 등록";
        } else {
            // 2. id가 이미 존재하는 유저일 경우 - cmd 1302
            result.cmd = 1302;
            result.message = "이미 등록된 유저입니다";
        }
        res.send(result);
    })
})
```

Node.js 기반 서버에서 JSON 데이터를 송수신
bcrypt 라이브러리를 활용한 비밀번호 해시 처리
로그인과 회원 가입 등 상황에 맞게 쿼리문 전달

구조 및 설계_DB

AWS에서 제공하는 RDS를 이용
MySQL과 연동해 데이터 CRUD 수행





감사합니다